AWS Serverless Microservices with Patterns & Best Practices

→ Event-driven Serverless Microservices with using AWS Lambda, Api Gateway, Event Bridge, SQS, DynamoDB and CDK for IaC

Mehmet Ozkaya

AWS Event-driven Serverless Microservices

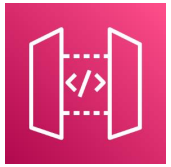
Compute

AWS Lambda

Databases

Amazon DynamoDB

API Management



Amazon API Gateway

Messaging/Application Integrations

Amazon EventBridge

Amazon Simple Queue Service (Amazon SQS)

IaC

AWS CloudFormation

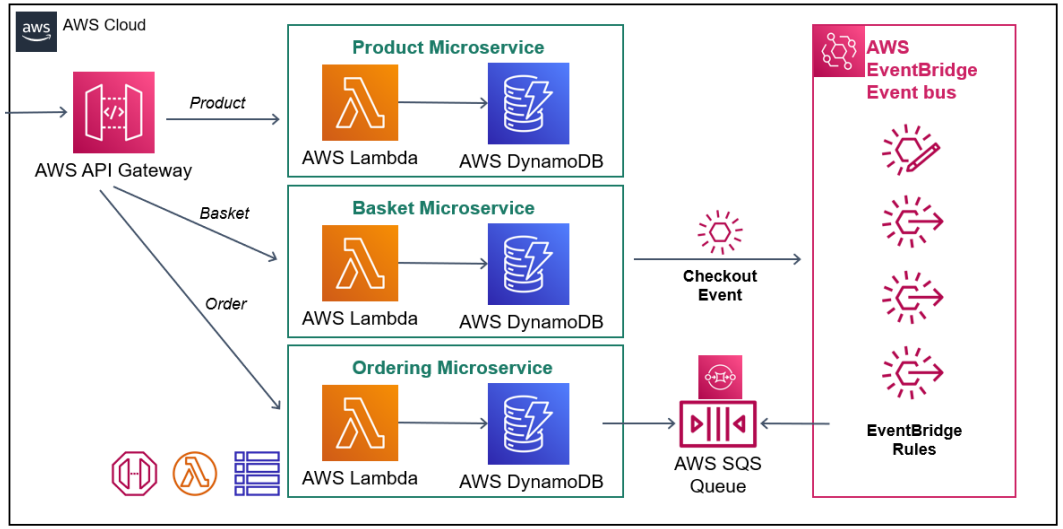
AWS CDK Stack

Monitoring

Amazon CloudWatch

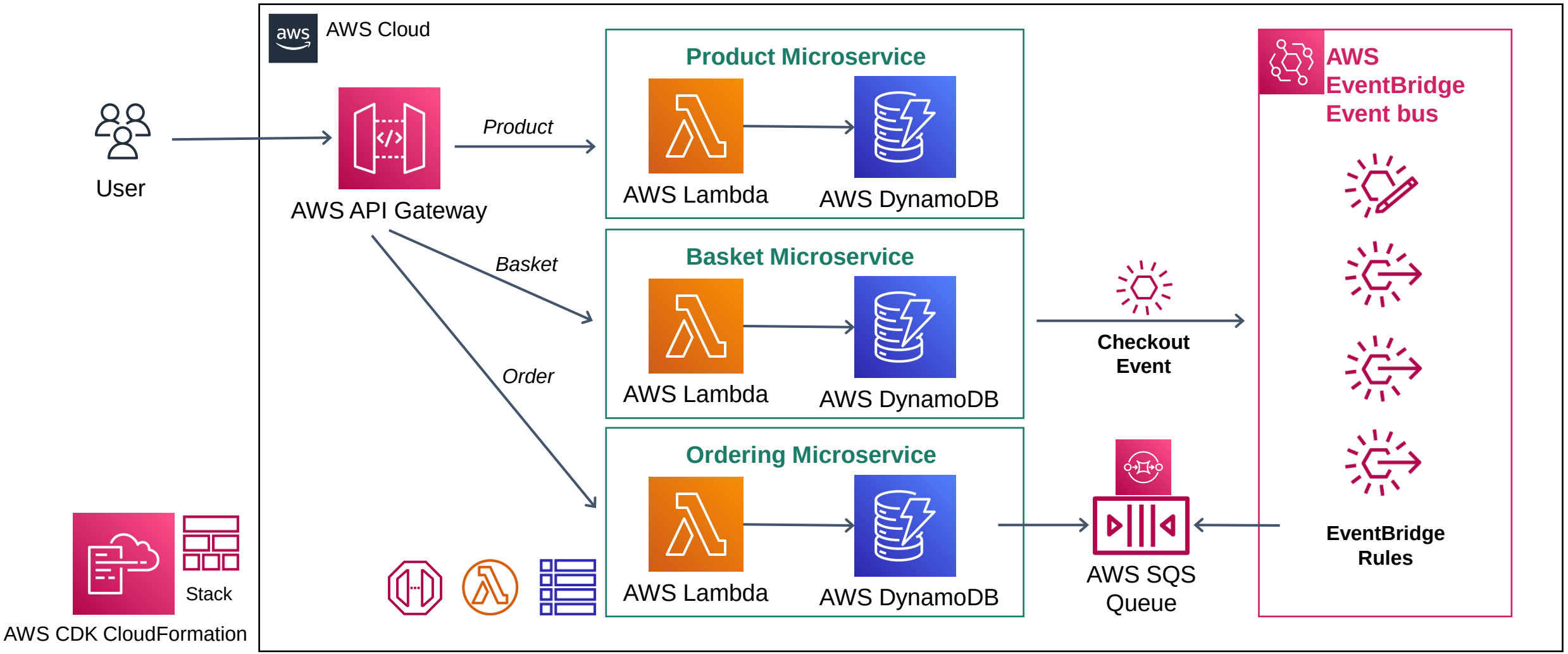
AWS Event-driven Serverless Microservices

- **REST API** and **CRUD** endpoints (AWS Lambda, API Gateway)
- **Data** persistence (AWS DynamoDB)
- **Decouple Microservices** with event (AWS EventBridge)
- **Message Queues** for cross-service communication (AWS SQS)
- **Cloud stack development** with IaC (AWS CloudFormation CDK)

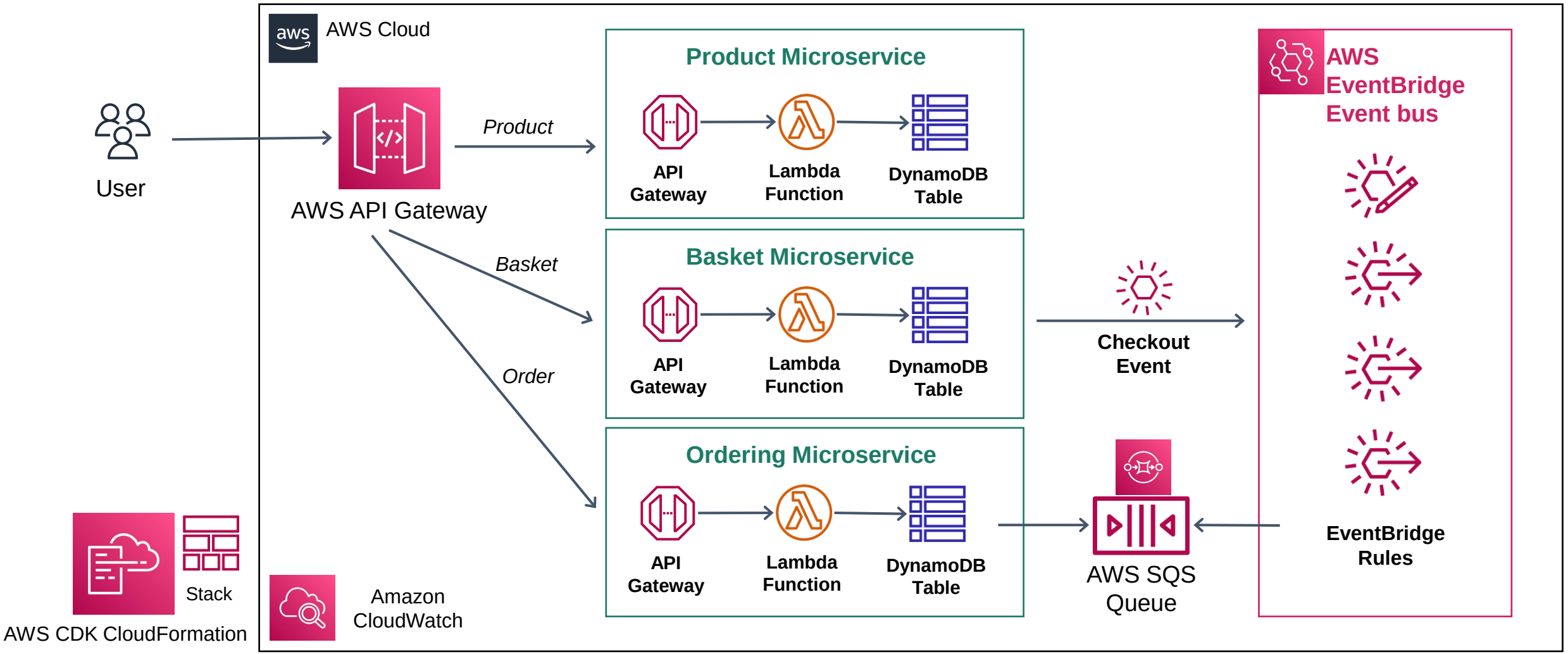


- **Event-driven** Serverless Microservices
- Follow the **Serverless Design Patterns** and **Best Practices**
- Developing **E-commerce Event-driven Microservices** application

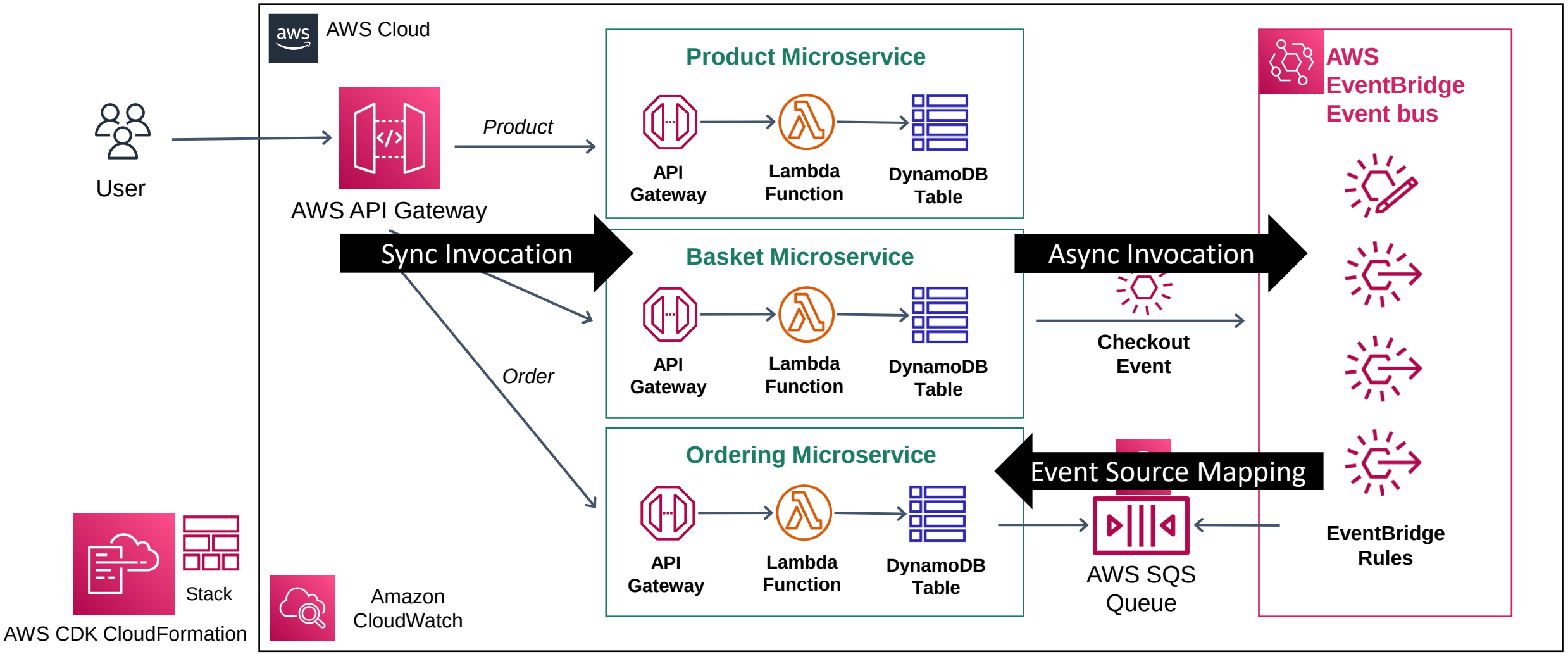
AWS Serverless Microservices for Ecommerce Application



AWS Serverless Microservices for Ecommerce Application

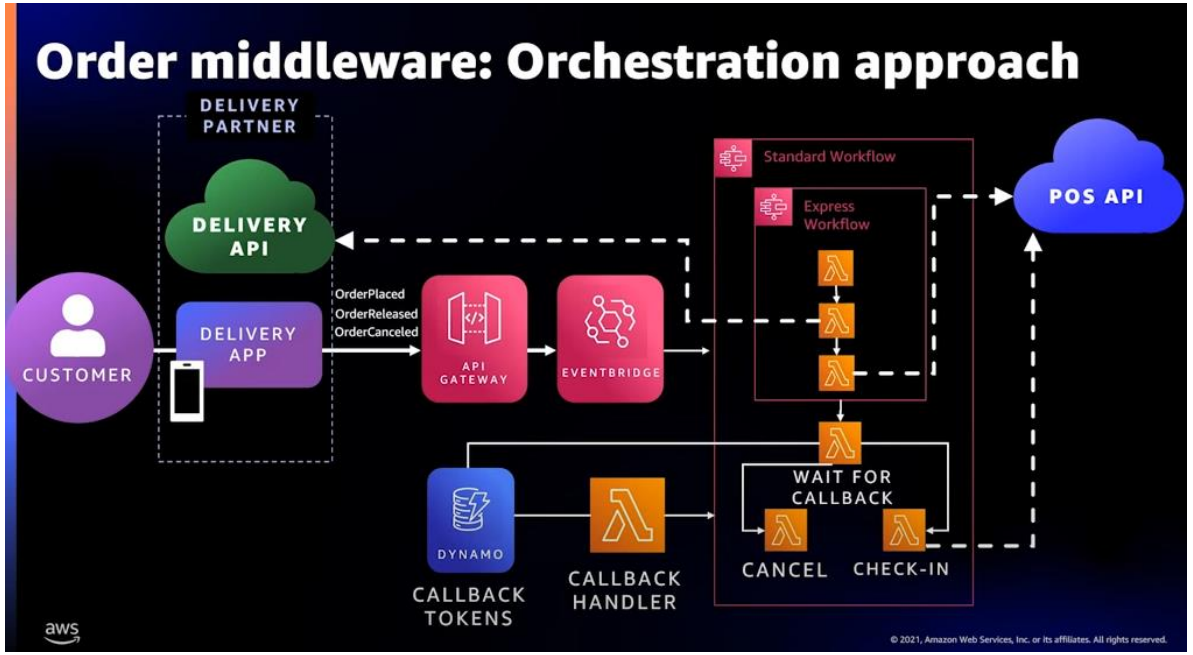


AWS Serverless Microservices for Ecommerce Application



AWS Developer and AWS Solution Architecture Jobs

- **AWS Developer** and **AWS Solution Architecture** jobs
- **3 figure** salaries
- **Real-world** Serverless Application
- **Taco Bell** is fully **serverless food delivery** company
- Follow **same development steps** of this kind of Serverless applications on AWS
- Get **hands-on experience**
- Demand for **AWS jobs** is continuously on the rise



https://www.youtube.com/watch?v=U5GZNt0iMZY

AWS Certifications

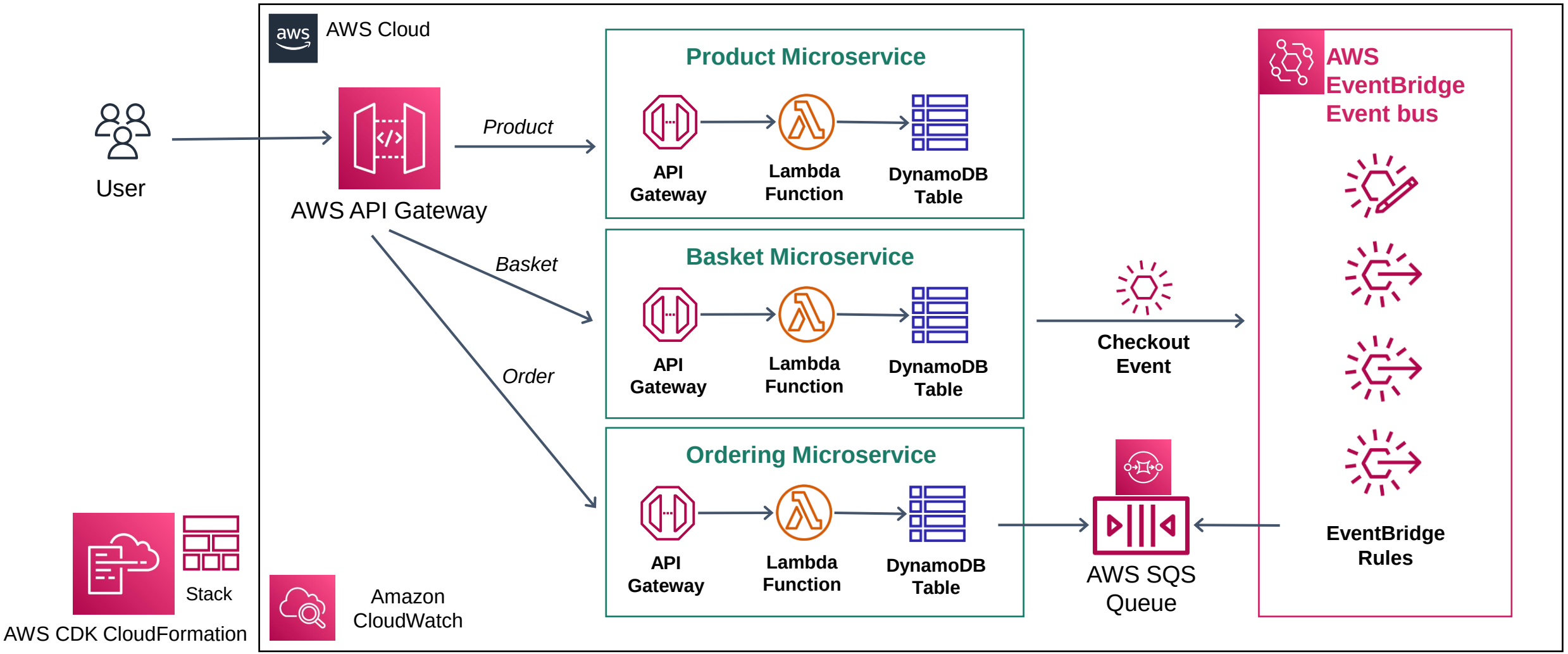
- **AWS Certified Developer Associate**
- **AWS Certified Solutions Architect Associate**
- Don't memorize all topics, feel and develop by hands-on, dirty your hands
- **Theorical** and mostly **practical** way with developing serverless e-commerce application

Serverless + CDK Automation + Integration Patterns = AWSome!

- **AWS Serverless** will be our application development framework.
- **AWS CDK** is IaC tool that we will develop whole infrastructure with typescript coding
- **Integration Patterns** with Queue-Chaning, Publish-Subscribe and Fan-out design patterns
- **AWSome** Microservices !

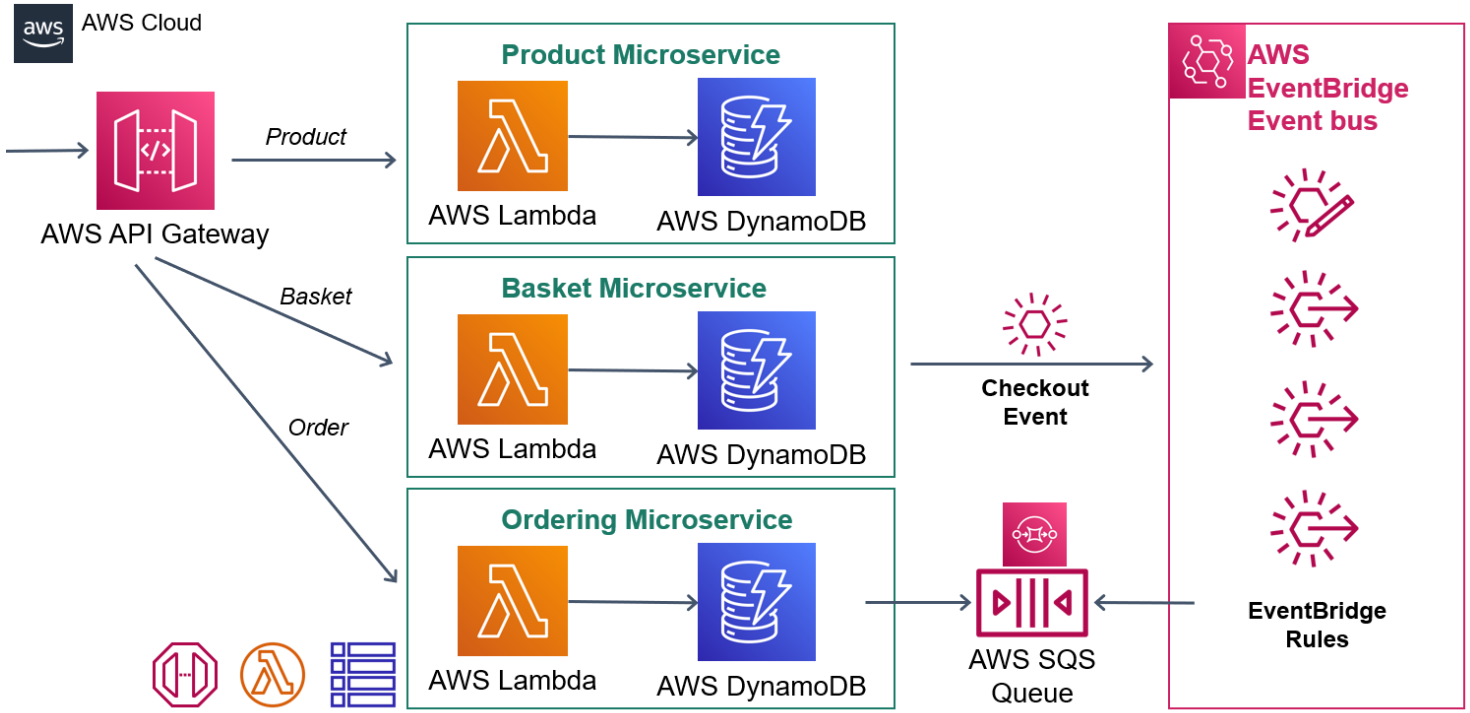
https://www.youtube.com/watch?v=ttJAIQf7cTw

AWS Serverless Microservices for Ecommerce Application

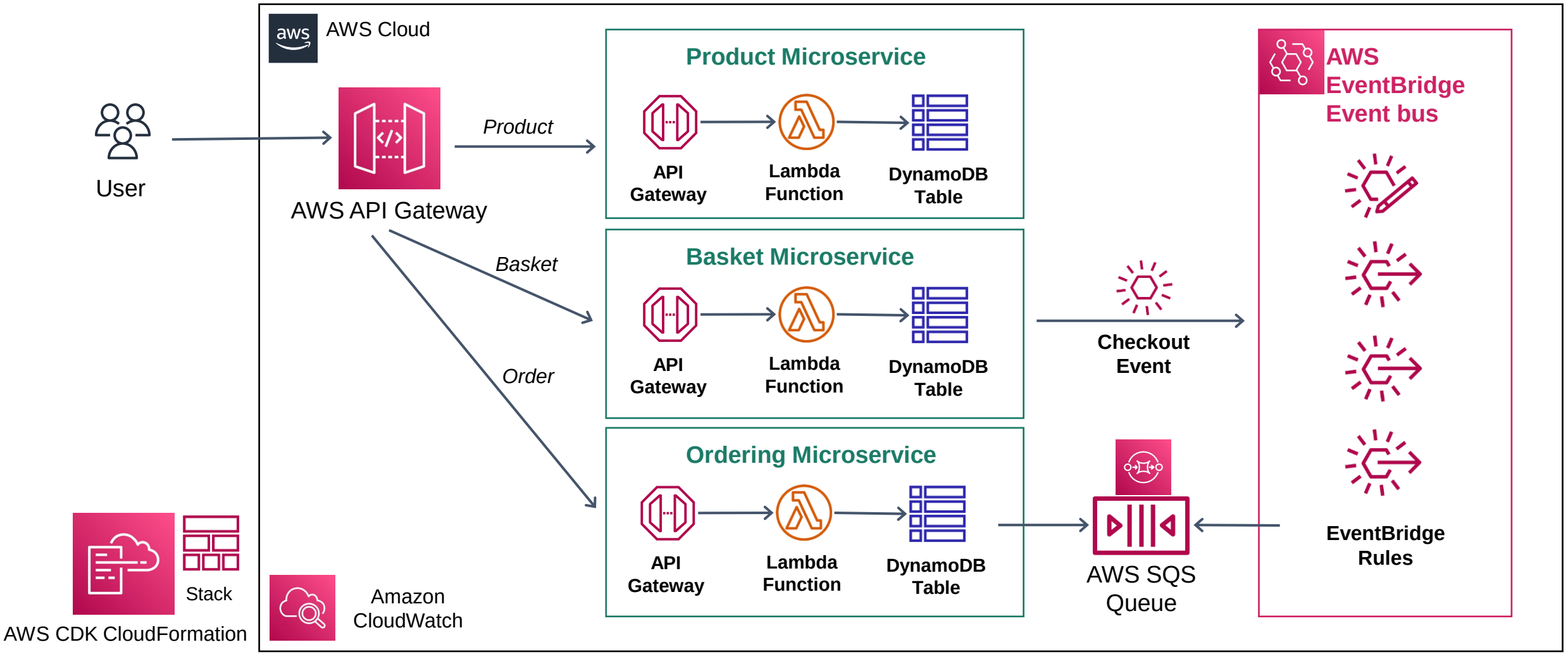


Run the Final Application – Serverless E-Commerce App

- Using **AWS CDK** for provision all architecture.
- Only **one** cdk command provision full architecture.
- Run **cdk deploy** command on Project folder at Visual Studio.

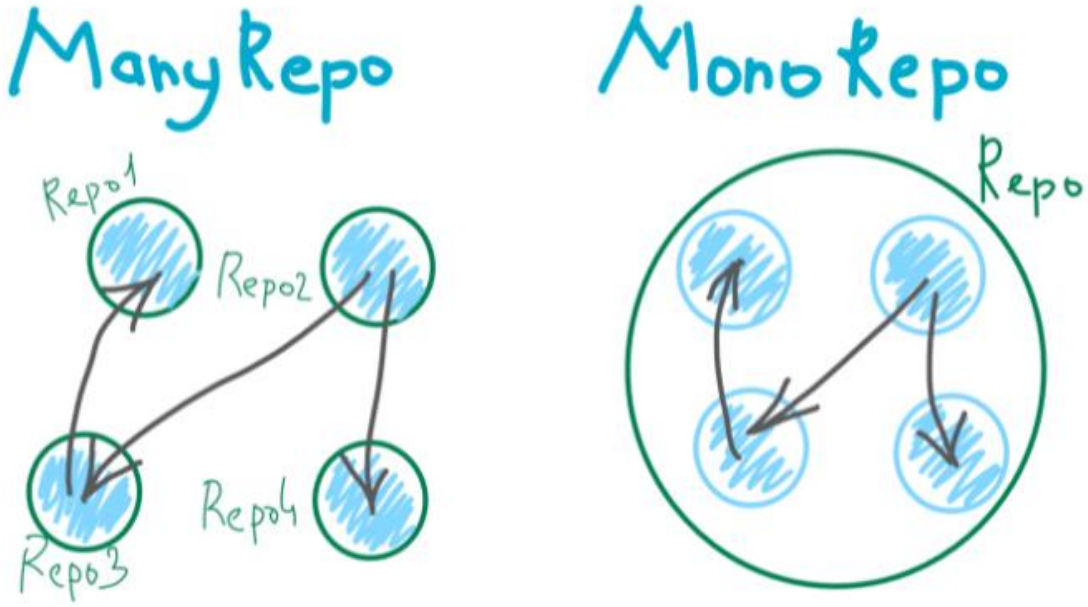


AWS Serverless Microservices for Ecommerce Application



What is Monorepo ?

- Software development strategy where many project codes are **stored** in the **same repository**
- **Single repository** that stores all of your code and assets for every project
- Companies like Google, Facebook, Microsoft, Uber, Airbnb, and Twitter
- **Single source of truth**, easy to **share code**, easy to refactor code
- Why We are using **Monorepo** ?
  - Microservices Part
  - Infrastructure Part
- Store **Microservices Code** and **Infrastructure Code** at the same repository.



https://medium.com/@jvr572/how-deploy-from-a-git-monorepo-1a9a55b23d44

2 Main Parts of Monorepo - Serverless E-commerce App

1

Serverless Infrastructure Development

IaC with AWS CDK using Typescript language

2

Microservices Lambda Function Development

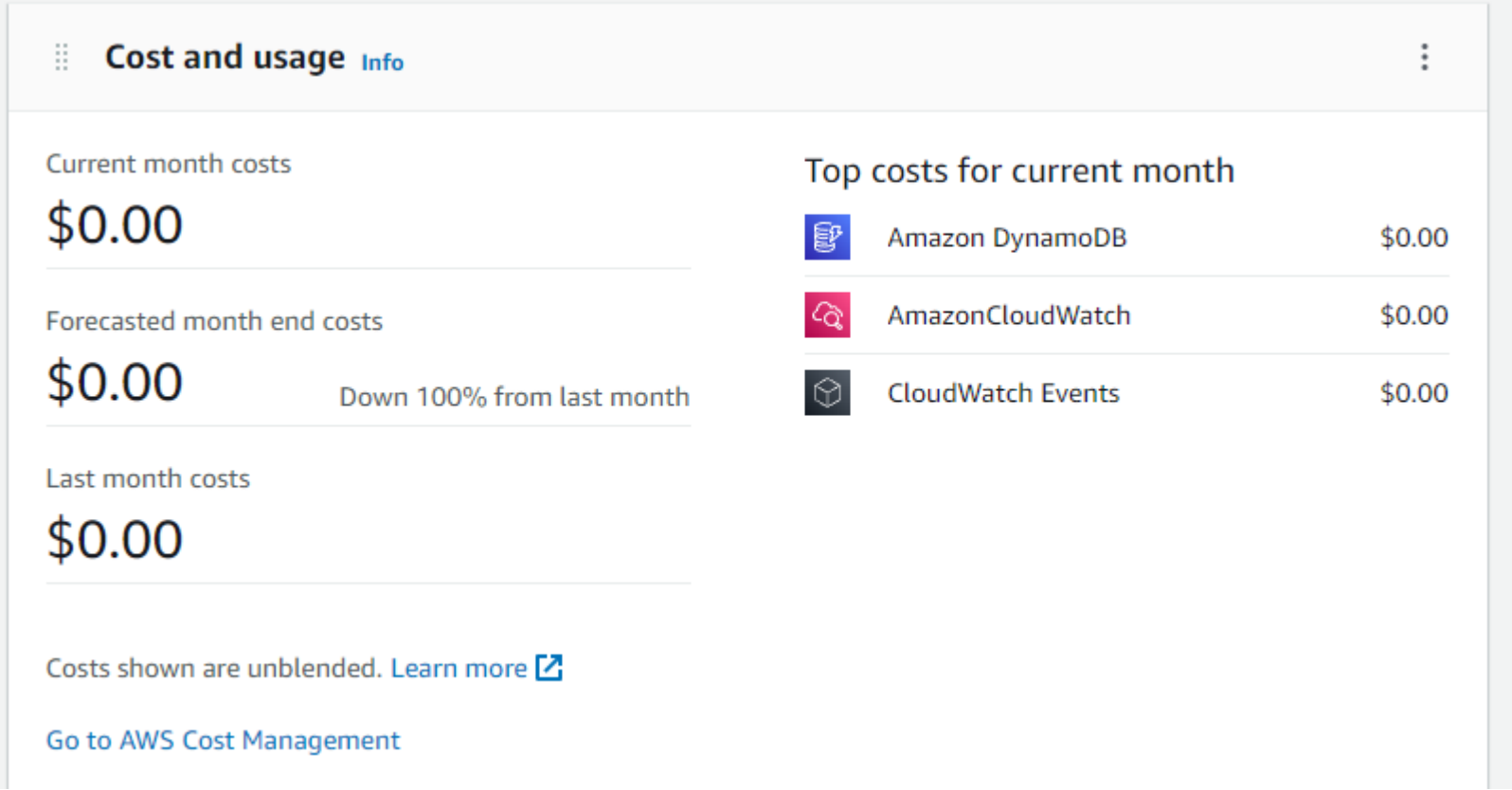
Nodejs Lambda Functions using AWS SDK for JavaScript v3

Project Folder Structure

- **Main Folders**
  bin, lib and src folders. bin/lib folders generate by aws cdk project template.
- **bin folder**
  Starting point of our application.
- **lib folder**
  Infrastructure codes. IaC Serverless Stacks with aws cdk.
- **src folder**
  Microservices development codes with nodejs.
- **DEMO** – Review Project Folder Structure into Visual Studio Code

AWS Cost Management During the Course

- Deployed full serverless e-commerce application with the **free tier account** on aws cloud
- AWS **didn't charge me** any amount of money from AWS **with free tier account**



AWS Usage During the Course

AWS Free Tier Info

Summary (9)

Find service name

1

Service	AWS Free Tier usage limit	Current usage	Forecasted usage	MTD actual usage %	MTD forecasted usage %
Amazon Simple Storage Service	2,000 Put, Copy, Post or List Requests of Amazon S3	153 Requests	198 Requests	7.65%	9.88%
Amazon Simple Storage Service	20,000 Get Requests of Amazon S3	137 Requests	177 Requests	0.69%	0.88%
AWS Key Management Service	20,000 free requests per month for AWS Key Management Service	133 Requests	172 Requests	0.66%	0.86%
Amazon Simple Storage Service	5 GB of Amazon S3 standard storage	0 GB-Mo	0 GB-Mo	0.04%	0.06%
AWS Lambda	1,000,000 free requests per month for AWS Lambda	216 Requests	279 Requests	0.02%	0.03%
Amazon API Gateway	1 Million API Calls per month of Amazon API Gateway	201 AmazonApiGatewayRequest	260 AmazonApiGatewayRequest	0.02%	0.03%
AmazonCloudWatch	5 GB of Log Data Ingestion for Amazon Cloudwatch	0 GB	0 GB	0.02%	0.02%
AmazonCloudWatch	5 GB of Log Data Archive for Amazon Cloudwatch	0 GB-Mo	0 GB-Mo	0.00%	0.00%
AWS Lambda	400,000 seconds of compute time per month for AWS Lambda	7 seconds	9 seconds	0.00%	0.00%

Turn off the lights before leaving the room

- **Amazon CTO Werner Vogels**
  Turn off the lights before leaving the room
- If you leave from computer and shut the light off, that means you also **shut down the AWS Resources**
- Think that we are going to prepare **demo session** with **your customer**
- Follow the best practices; **clear all resources** after you finished the work at that day
- **Don’t worry**, when you come back tomorrow, we can instantly come the same situation with the power of **AWS CDK**

https://www.youtube.com/watch?v=8_Xs8Ik0h1w

Don’t Share AWS Account / Sub User / Api Access Key

- Don’t Share
  - AWS Account
  - Sub User
  - Api Access Key
- Avoid **unexpected cost** from AWS
- If you share, that means you **allow** to someone **create resource** on AWS
- Follow these principles
  - Turn off the lights before leaving the room
  - Don’t Share AWS Account / Sub User / Api Access Key

Course Structure and Way of Learning

1

Exploring AWS Service

Learn Theoretical Information

AWS Console WalkThrough Demo

2

Hands-on Development

Infrastructure Development with AWS CDK

Application Development with NodeJS using AWS SDK

Course Structure and Way of Learning – Exploring Part

- **Part 1 - Learn Theorical Information**
  This is Overview and detail theorical information of the topic.
- For example if we talk about AWS SQS, we will see
  - SQS Overview
  - Core Concepts
  - Example Use Cases
  - Main Features like Visibility Timeout, Batch Processing, Type of SQS - FIFO, Standart
  - Best Practices of SQS
- **Part 2 - AWS Console WalkTrough Demo**
  Make a demo with Walkthrough of AWS Console. Map theorical information with the AWS Console.
- For example if we talk about AWS SQS, we will see
  - Create SQS on Console
  - Set visibility timeout
  - Send message to queue
  - Poll message to queue

Course Structure and Way of Learning – Development Part

- **Part 3 - Infrastructure Development with CDK**
  Serverless Infrastructure Development - [AWS CDK Typescript IaC]
- For example if we talk about AWS SQS, we will see
  - Developing AWS CDK typescript code for AWS SQS and provision it

- **Part 4 - Application Development with NodeJS using AWS SDK**
  Microservices Lambda Function Development - [Nodejs Lambda Functions using AWS SDK for JavaScript v3]
- For example if we talk about AWS SQS in our serverless e-commerce application, we will see
  - Developing Ordering Microservice poll event queue business logic with AWS SDK

Example of AWS SQS Way of Learning

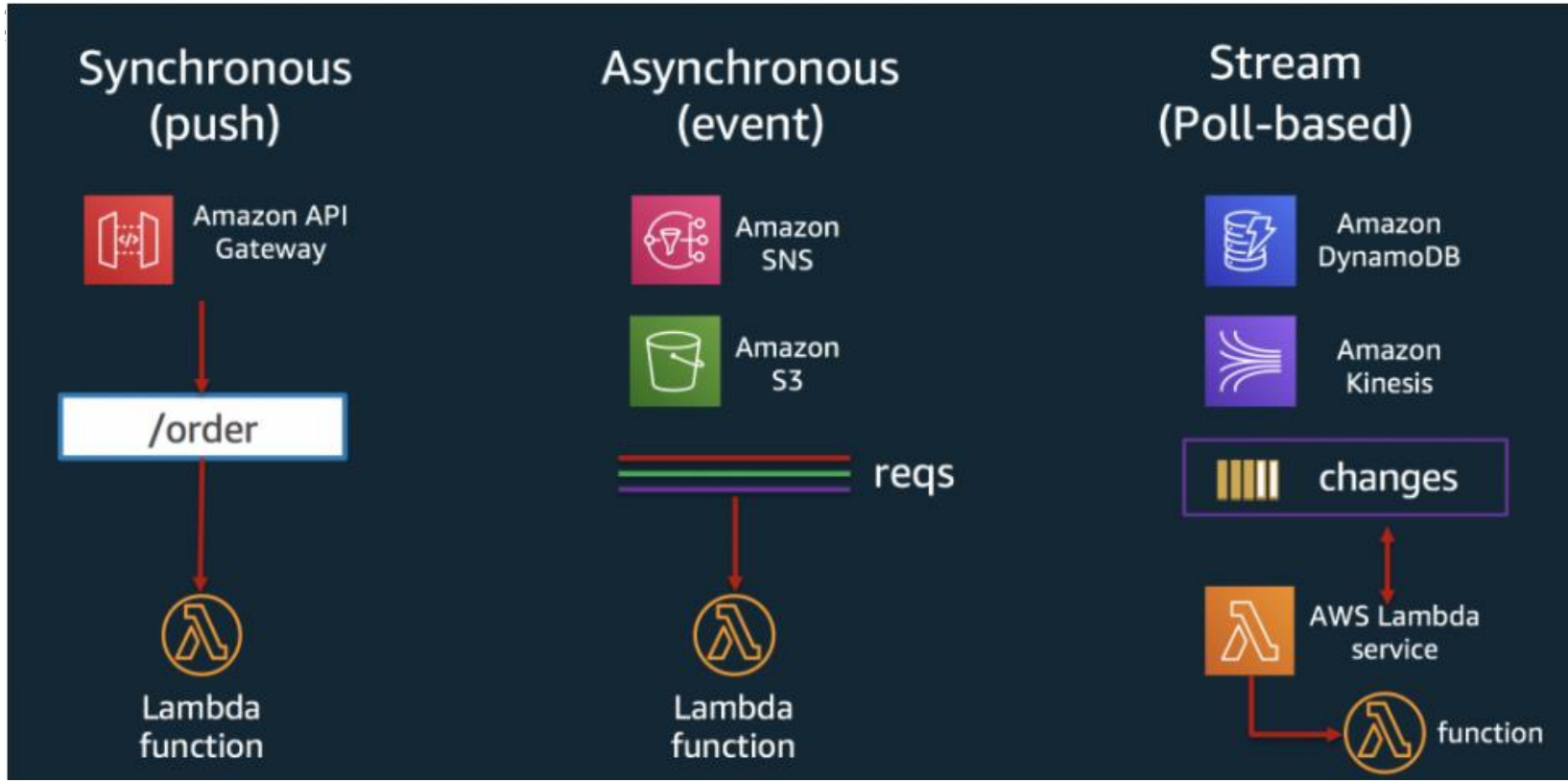
- So that mean we will follow these steps :
- Part 1 - AWS SQS Overview
- Part 2 - AWS SQS Walk Trough Step by Step Tutorial over Console
- Part 3 - AWS SQS Infrastructure as Code Development with AWS CDK
- Part 4 - Developing AWS SQS Interaction with Ordering Microservices using AWS SDK

- Same way Lambda Microservices, DynamoDb, EventBridge and so on..
- I will explain every step of developments of our serverless e-commerce application.

AWS Lambda Invocation Types

- Triggered lambda functions with different AWS Lambda Invocation Types
- AWS Lambda has 3 Invocation Types;
- **Lambda Synchronous invocation**
- **Lambda Asynchronous invocation**
- **Lambda Event Source Mapping**

with polling invocation



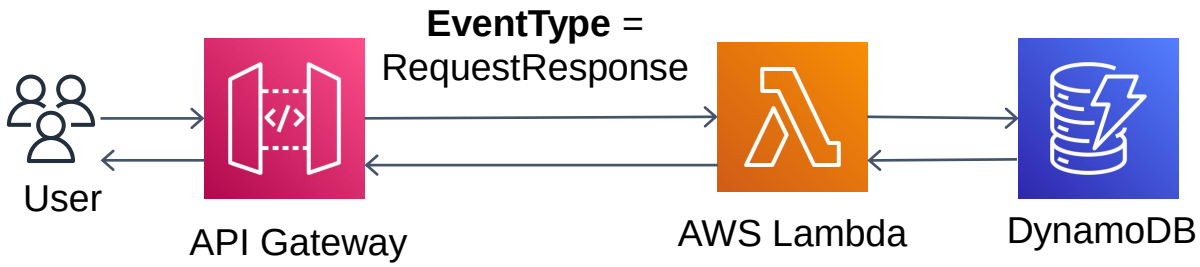
https://aws.amazon.com/blogs/architecture/understanding-the-different-ways-to-invoke-lambda-functions/

AWS Lambda Synchronous Invocation

- **Execute immediately** when you perform the Lambda Invoke API call.
- **Wait** for the function to process the function and return back to response.
- **API Gateway + Lambda + DynamoDB**
- **Invocation-type** flag should be **“RequestResponse”**
- Responsible for **inspecting** the **response**

  and **determining** if there was an **error**

  and **decide** to **retry** the invocation



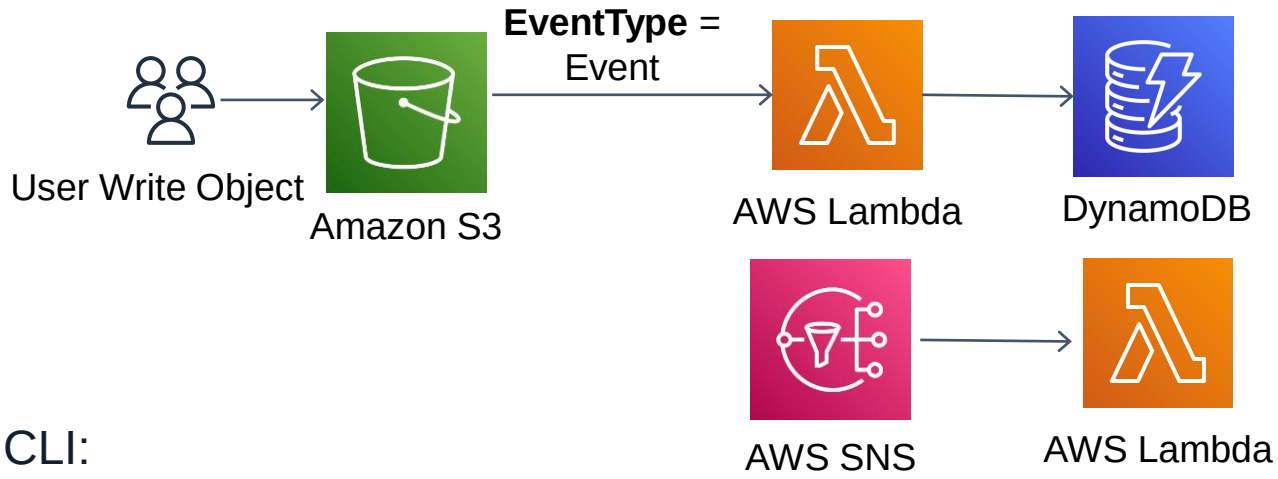
- Example of synchronous invocation using the AWS CLI:
  aws lambda invoke —function-name MyLambdaFunction **—invocation-type RequestResponse** —payload '{ "key": "value" }’
- Triggered **AWS services** of **synchronous invocation**; ELB (Application Load Balancer), Cognito, Lex, Alexa, API Gateway, CloudFront, Kinesis Data Firehose

AWS Lambda Asynchronous Invocation

- Lambda **sends** the **event** to a **internal queue** and returns a **success response** without any additional information
- Separate process **reads events** from the **queue** and **runs** our lambda function
- **S3 / SNS + Lambda + DynamoDB**
- **Invocation-type** flag should be **“Event”**
- AWS Lambda sets a **retry policy**

  Retry Count = 2

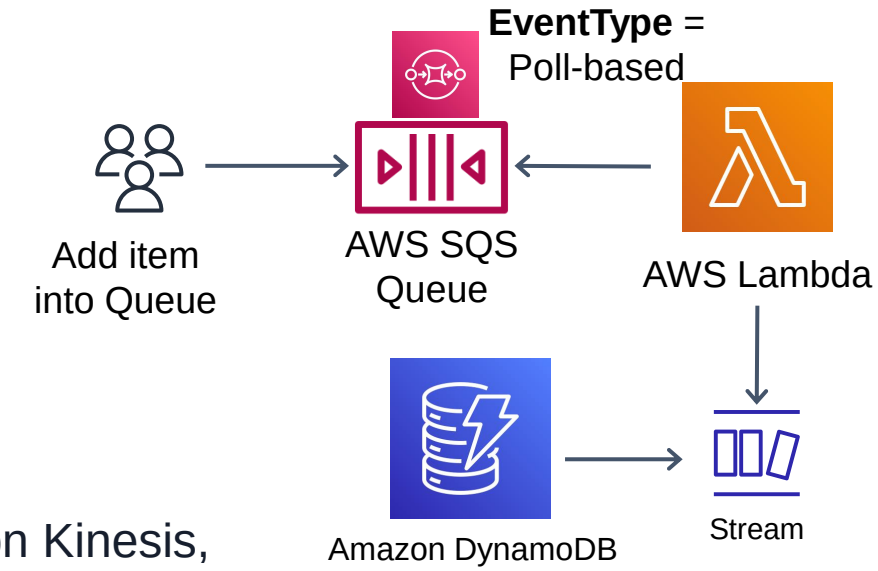
  Attach a Dead-Letter Queue (DLQ)



- Example of asynchronous invocation using the AWS CLI:
  aws lambda invoke —function-name MyLambdaFunction **—invocation-type Event** —payload '{ "key": "value" }'
- Triggered **AWS services** of **asynchronous invocation**; S3, EventBridge, SNS, SES, CloudFormation, CloudWatch Logs, CloudWatch Events, CodeCommit

AWS Lambda Event Source Mapping with Polling Invocation

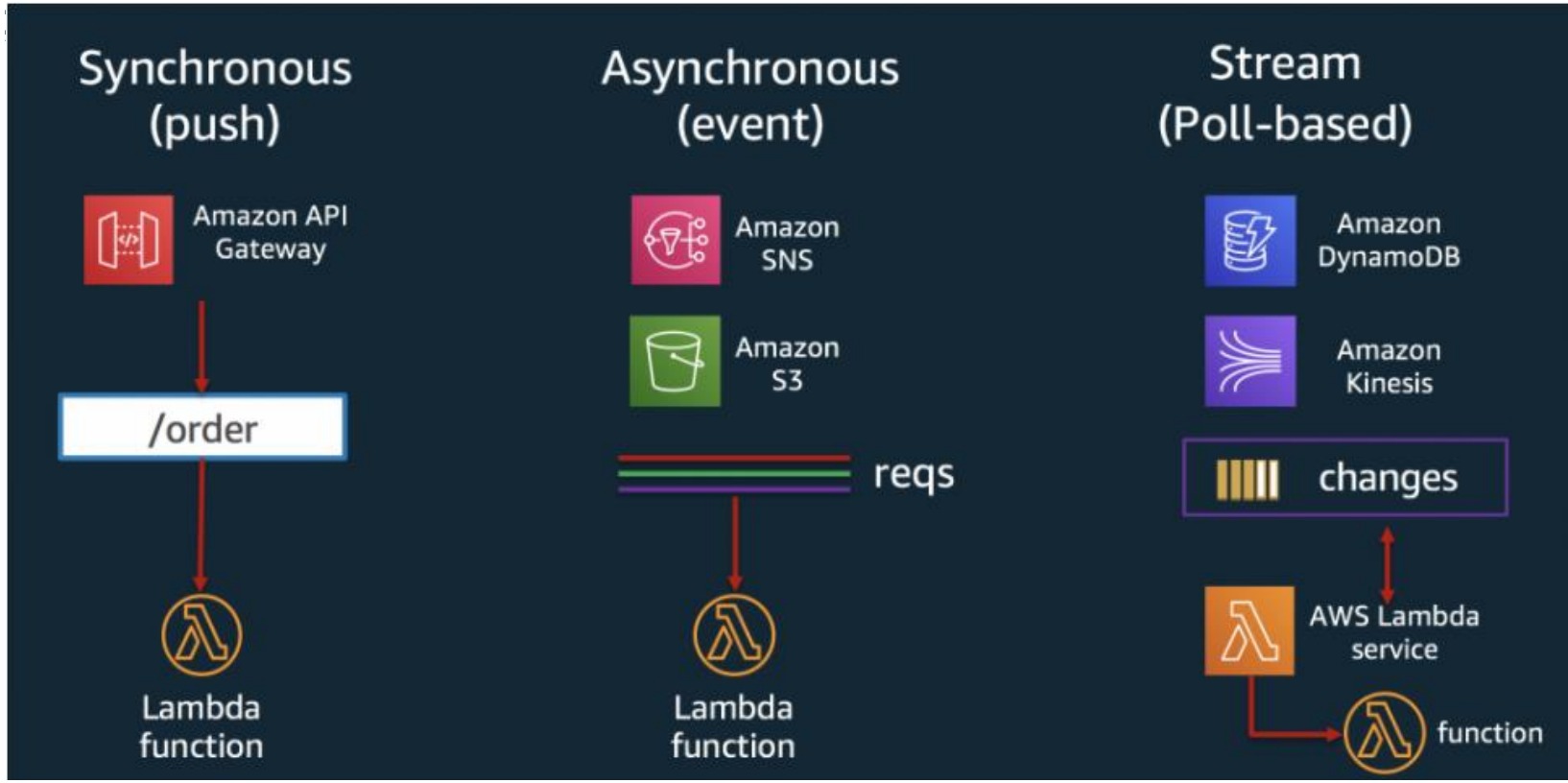
- **Pool-Based invocation** model allows us to **integrate** with **AWS Stream** and **Queue based services**.
- Lambda will **poll** from the AWS SQS or Kinesis streams, retrieve records, and invoke functions.
- Data stream or queue are **read** in **batches**,
- The function receives multiple items when execute function.
- **Batch sizes** can configure according to service types
- **SQS + Lambda**
- **Stream based processing** with **DynamoDB Streams + Lambda**
- Triggered **AWS services** of **Event Source Mapping invocation**; Amazon Kinesis, DynamoDB, Simple Queue Service (SQS)



AWS Lambda Invocation Types

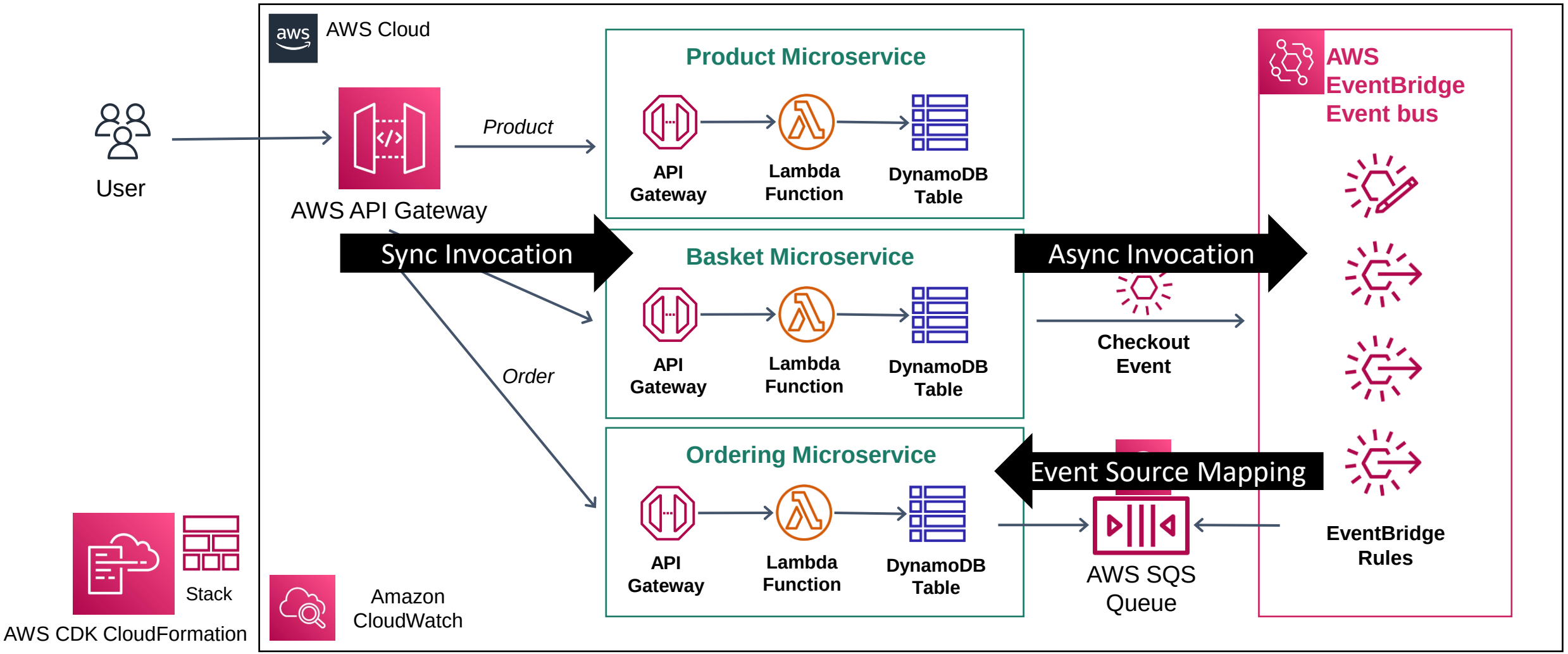
- Triggered lambda functions with different AWS Lambda Invocation Types
- AWS Lambda has 3 Invocation Types;
- **Lambda Synchronous invocation**
- **Lambda Asynchronous invocation**
- **Lambda Event Source Mapping**

with polling invocation

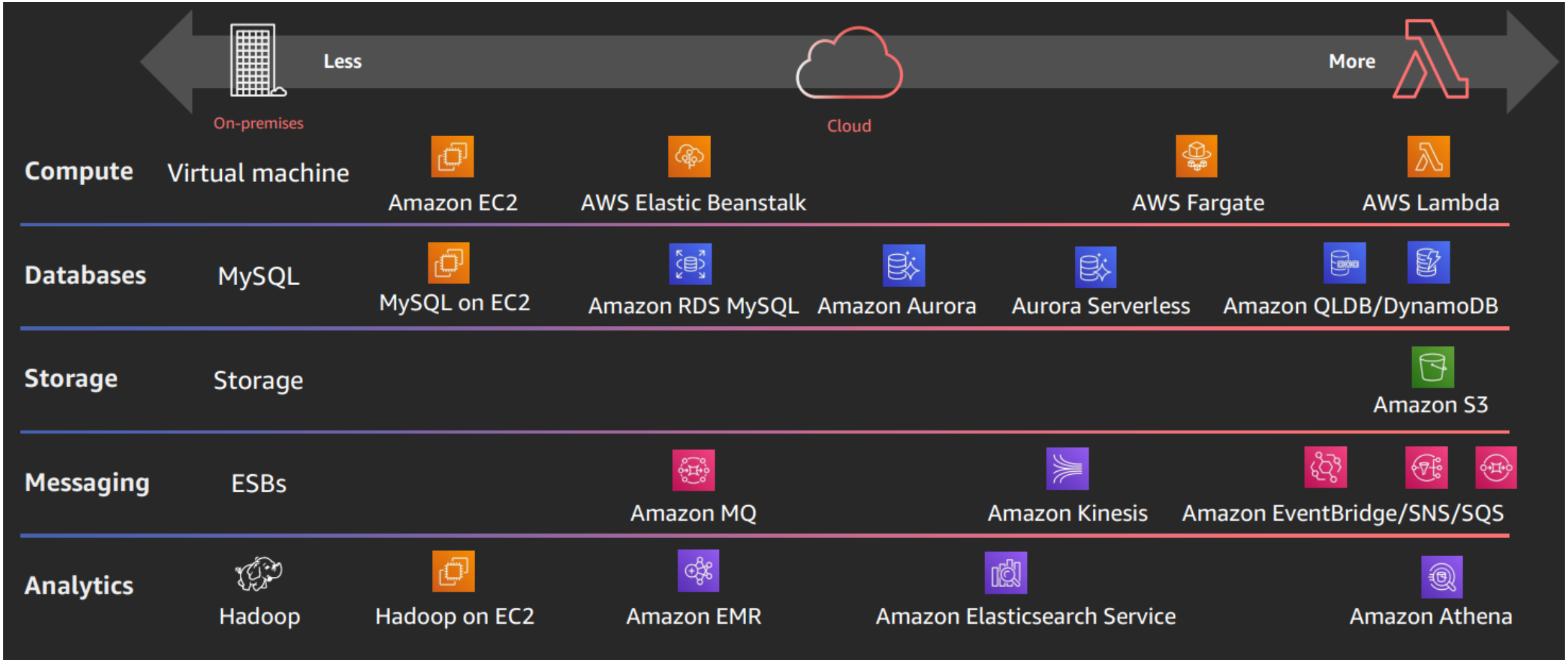


https://aws.amazon.com/blogs/architecture/understanding-the-different-ways-to-invoke-lambda-functions/

AWS Serverless Microservices with AWS Lambda Invocation Types



Serverless Explanained : AWS Operational Responsibility Model



https://d1.awsstatic.com/events/reinvent/2019/REPEAT_3_Serverless_architectural_patterns_and_best_practices_ARC307-R3.pdf

AWS Serverless services used

Compute

AWS Lambda

Databases

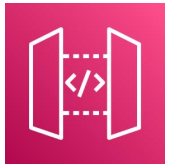
Amazon DynamoDB

Messaging/Application Integrations

Amazon EventBridge

Amazon Simple Queue Service (Amazon SQS)

API Management



Amazon API Gateway

IaC

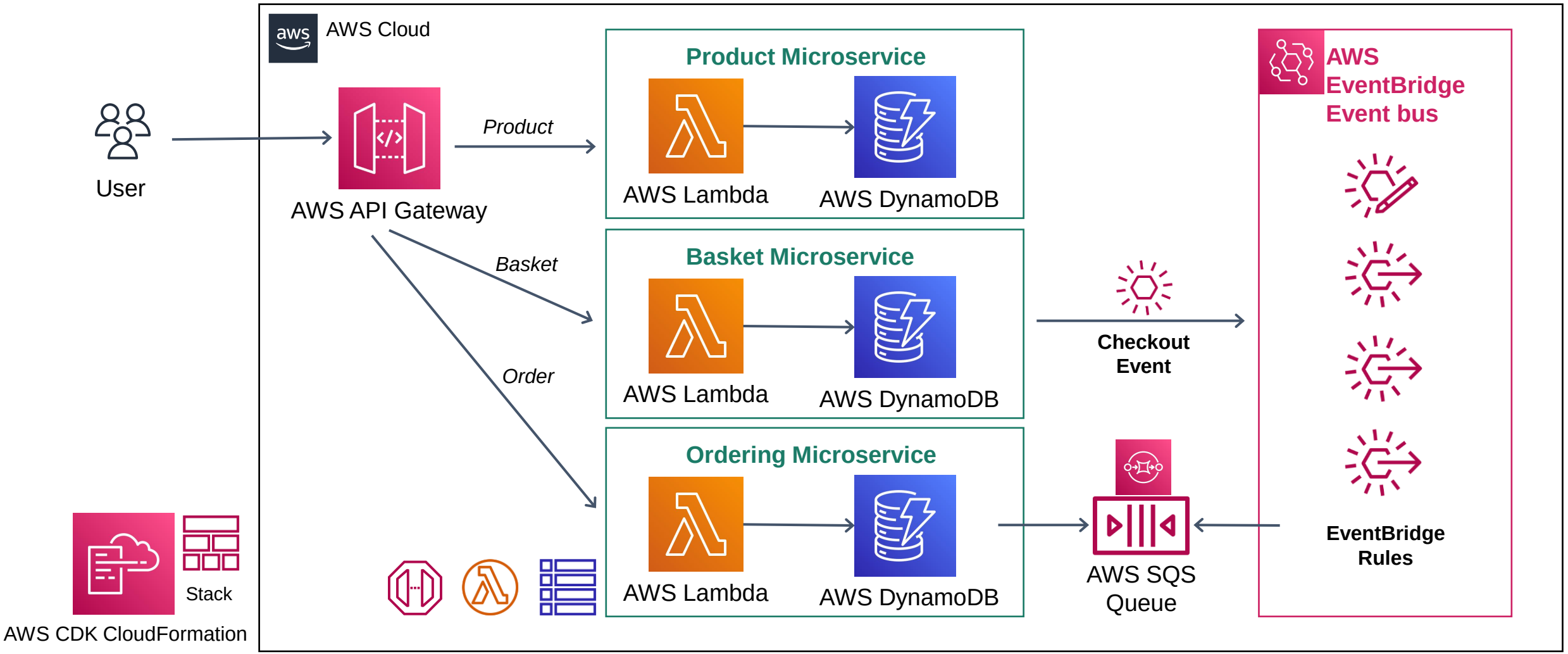
AWS CloudFormation

AWS CDK Stack

Monitoring

Amazon CloudWatch

AWS Serverless Microservices for Ecommerce Application



Project Code & Course Slides

→ Check our Serverless E-Commerce Microservices Application Project Code & Course Slides

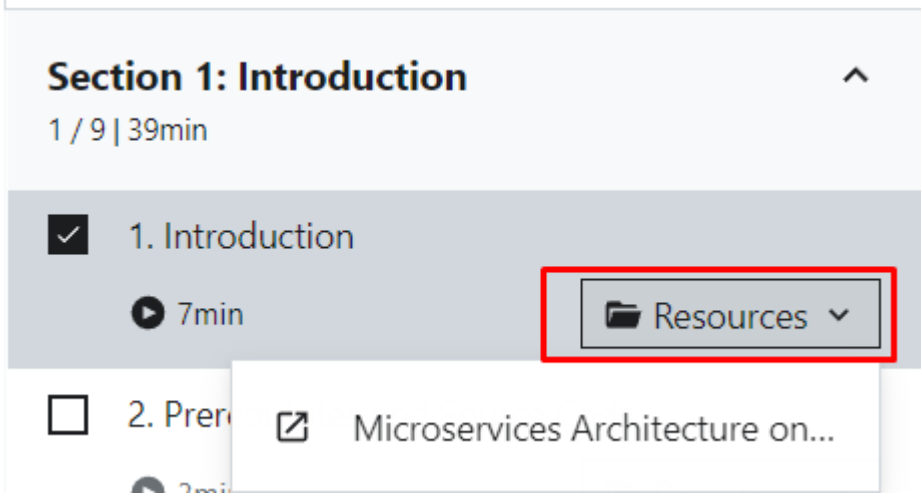
Project Code

- **Github Repository**
  Find full source code on Github repository
- awsrun organization and created aws-microservices repository
- **Section by Section Github Repository**
  Follow course section by section with aws-microservices-course-sections repository
- Clone or download both the repositories on GitHub
- Shared the links in the videos resources
- Ask questions from Q&A section

Course Slides

- **Powerpoint slides**
  Find full PowerPoint slides the link in the resource of this video.

- The official AWS icon set for building AWS architecture diagrams.
  - **https://aws.amazon.com/architecture/icons/**

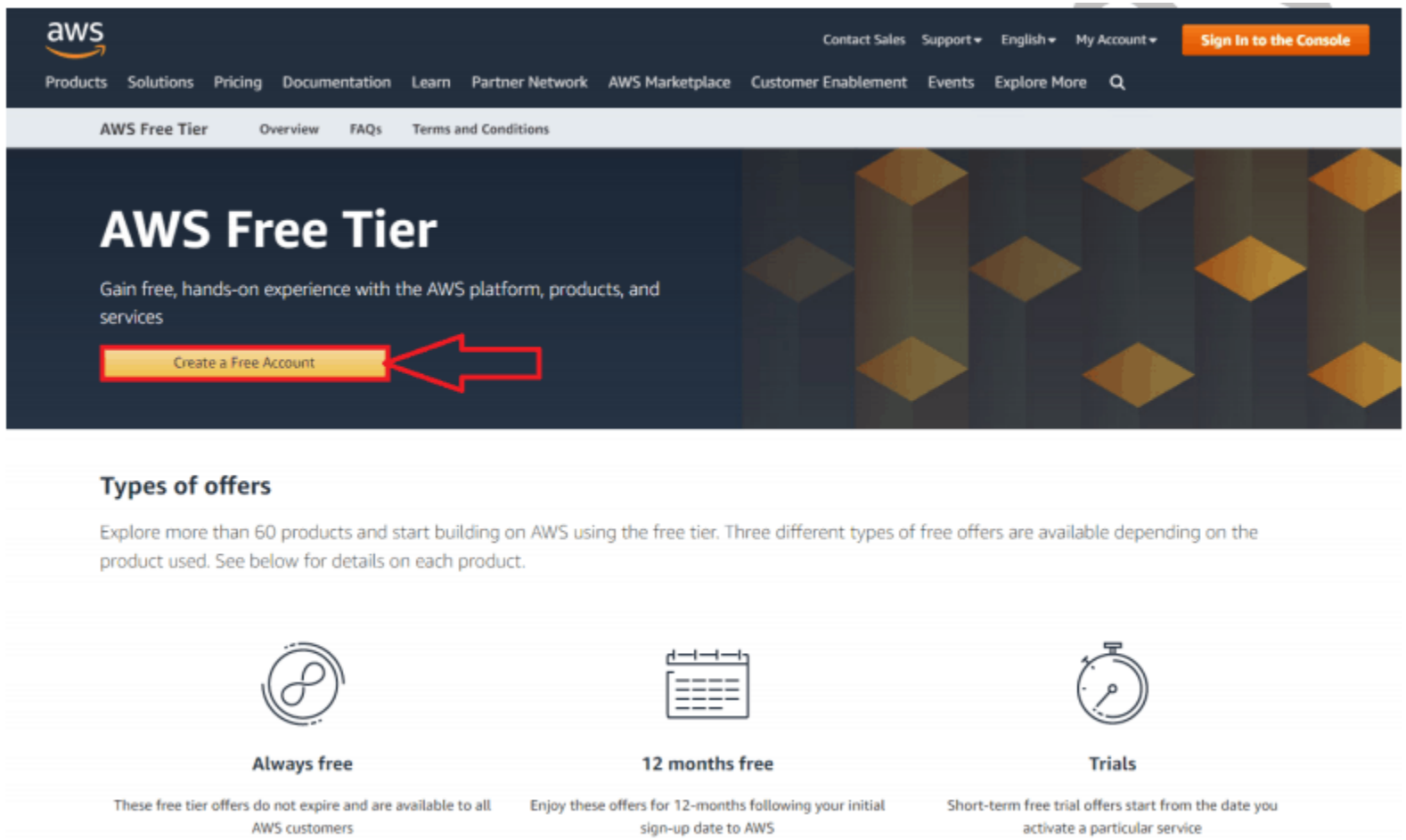


Create AWS Account - Free Tier

- Create our Free Tier AWS Account for creating resources during the course.

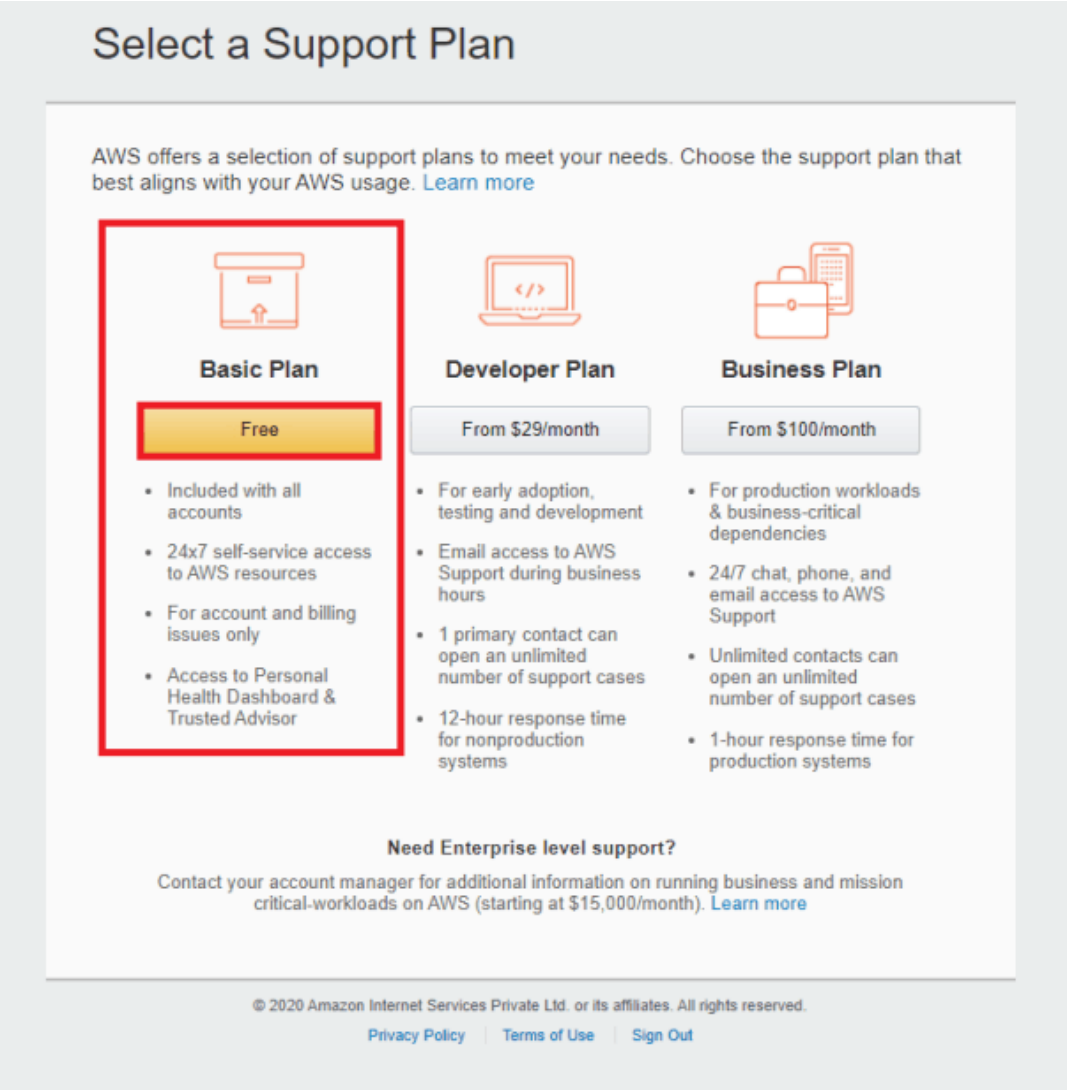
Create Free Tier AWS Account

- [Clik Here for AWS Create Free Tier Account]
- [Follow the Steps to Activate Free Tier Account]
- More than 100 products for building applications.
- Most of the services allowed good amount of usage depending on the product
- For example **25 GB DynamoDb, 1 million Lambda request, 1 TB Amazon CloudFront ..**



Select a Support Plan

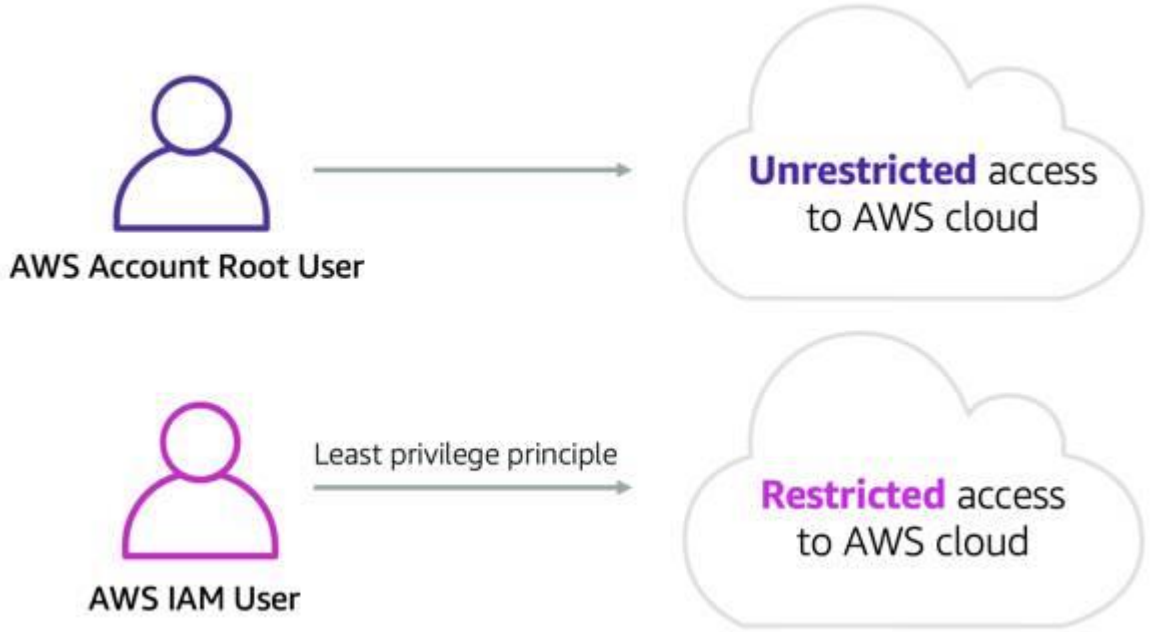
- AWS support offers a selection of plans to meet your business needs.
- Select **Basic Plan** support for free account
- created AWS Free Tier Account
- Wait for account activation
- If you face any problem during this process, you can [follow this page]



Security Best Practices of AWS Accounts

- AWS account has **2 main user type**
  - **Root User Account**
  - **IAM User Account**

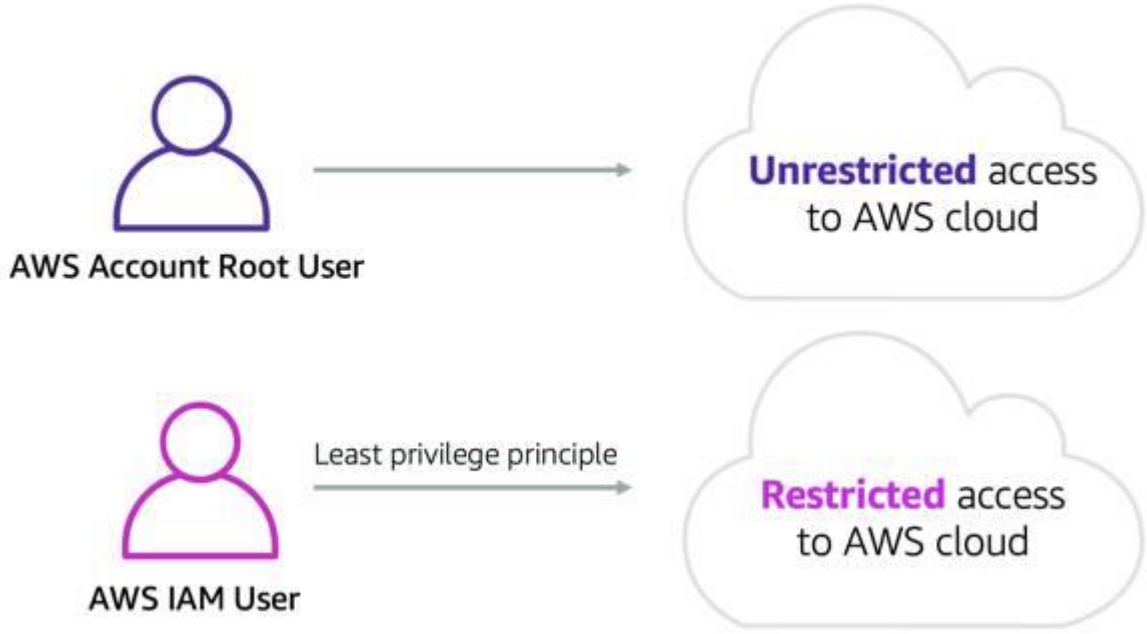
- **Root user account** has full power of your AWS account and it has **unrestricted access** over to your AWS cloud account
- **IAM User Account** is **sub-users** under the root user account and **define policies** over this account and **restrict** over to your AWS cloud account
- **Security Best Practice;** After active our account, first thing we should do create "**IAM User**" under root account for our **daily usage** of AWS Console
- [Security best practices in IAM]



https://trailhead.salesforce.com/en/content/learn/modules/aws-identity-and-access-management/set-iam-policies

Create IAM User Account and Configure for Programmatic and Console Access

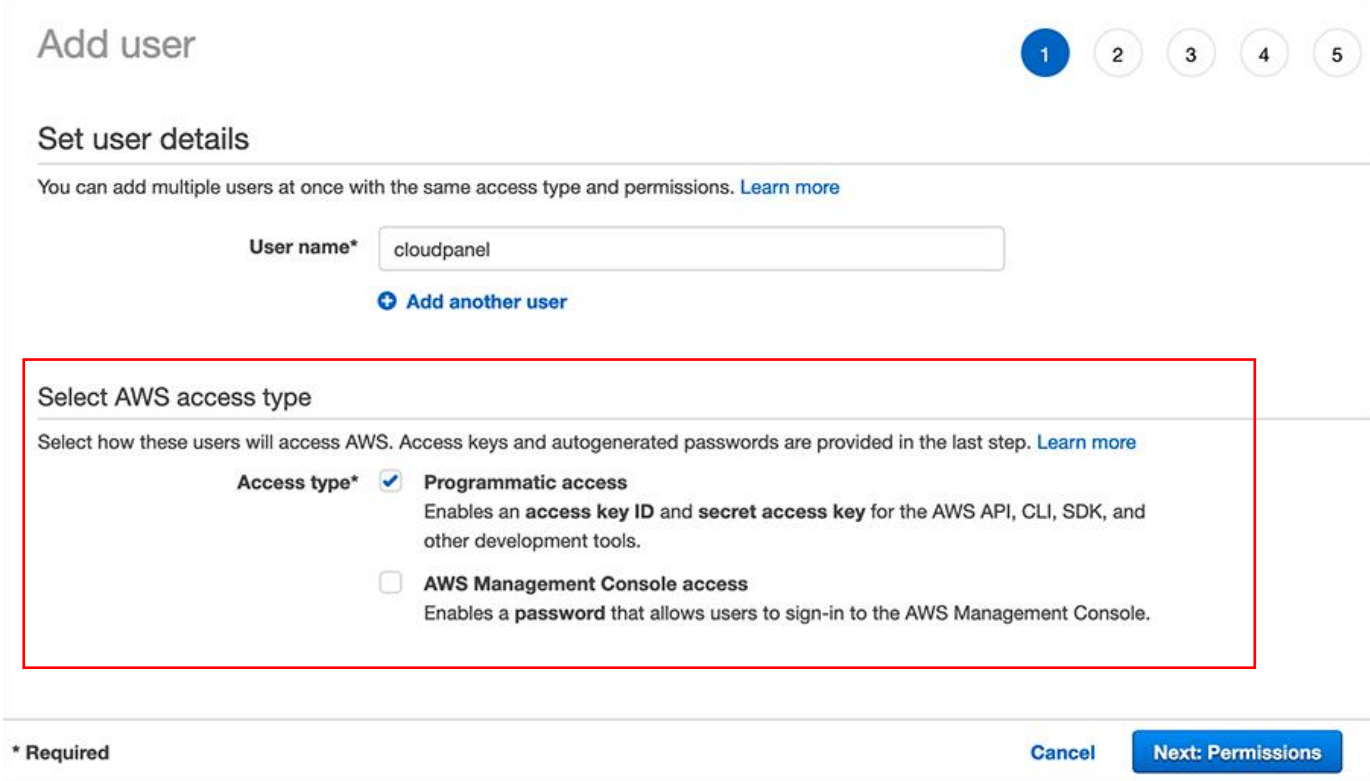
- Create **user-specific AWS account** under root user account that they have own login and passwords
- Define **Programmatic and Console Access**
- **Programmatic access is required** for our course, because we will use all interactions with AWS resources like **AWS Console, AWS CLI, AWS CDK and AWS SDK.**
- Follow me from AWS Console or follow article below;
- Create IAM User Account and Configure for Programmatic and Console Access
  - **Don’t forget to allow Programmatic Access**



https://trailhead.salesforce.com/en/content/learn/modules/aws-identity-and-access-management/set-iam-policies

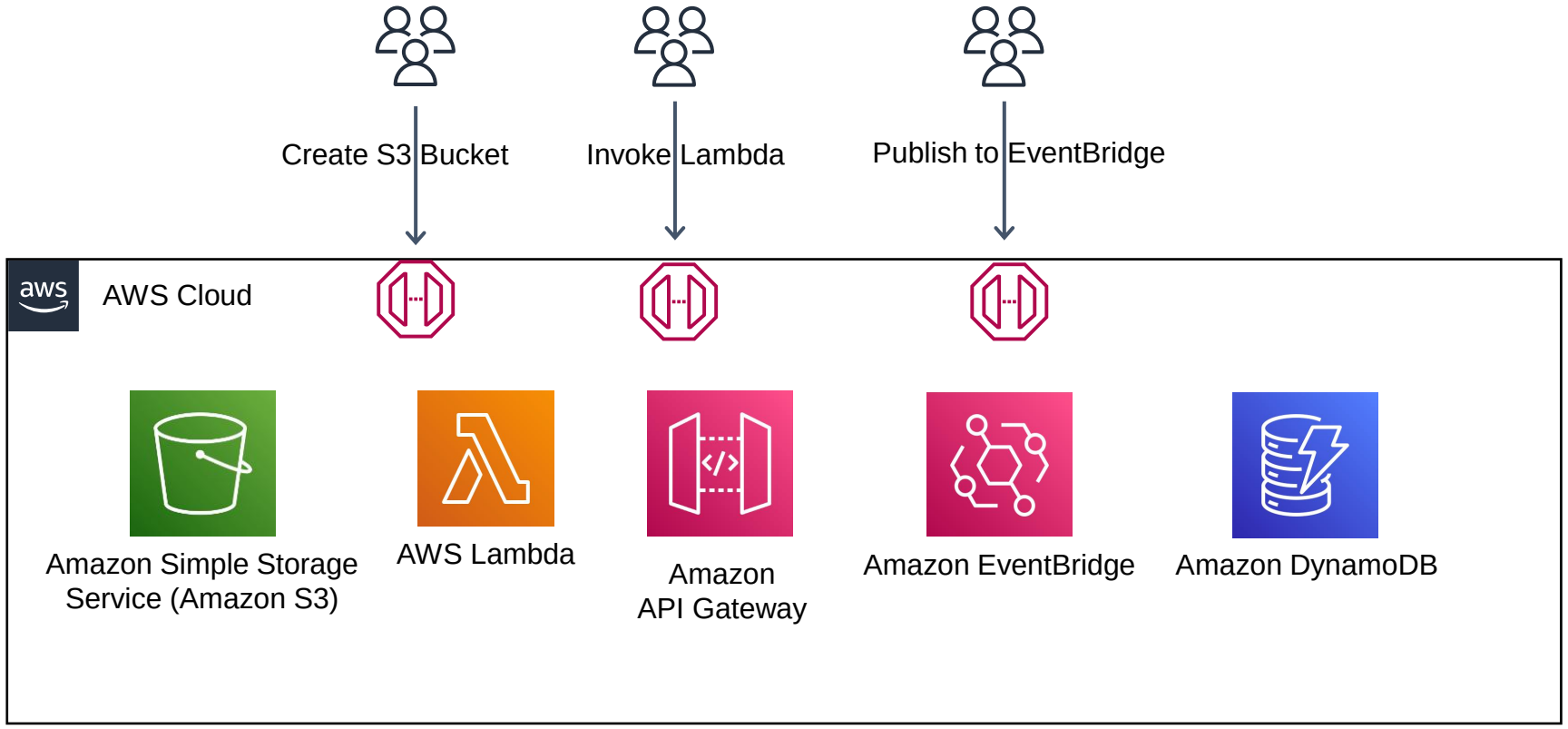
AWS Access Types - Programmatic / Management Console Access

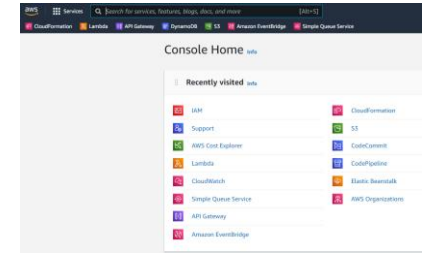
- When creating a user, we have an option about **AWS Access Types**
  - **Programmatic Access**
  - **AWS Management Console Access**
- **Programmatic Access**
  Enables **access key ID** and **secret access key** for the **AWS API, CLI, SDK,** and other development tools.
- During the course we will almost use all of these programmatic access types and of course use AWS Console every time



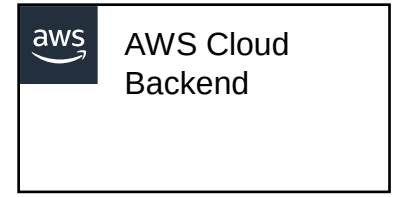
AWS Access with APIs

- **AWS expose APIs** that we can invoke to **create** and **manage aws services**



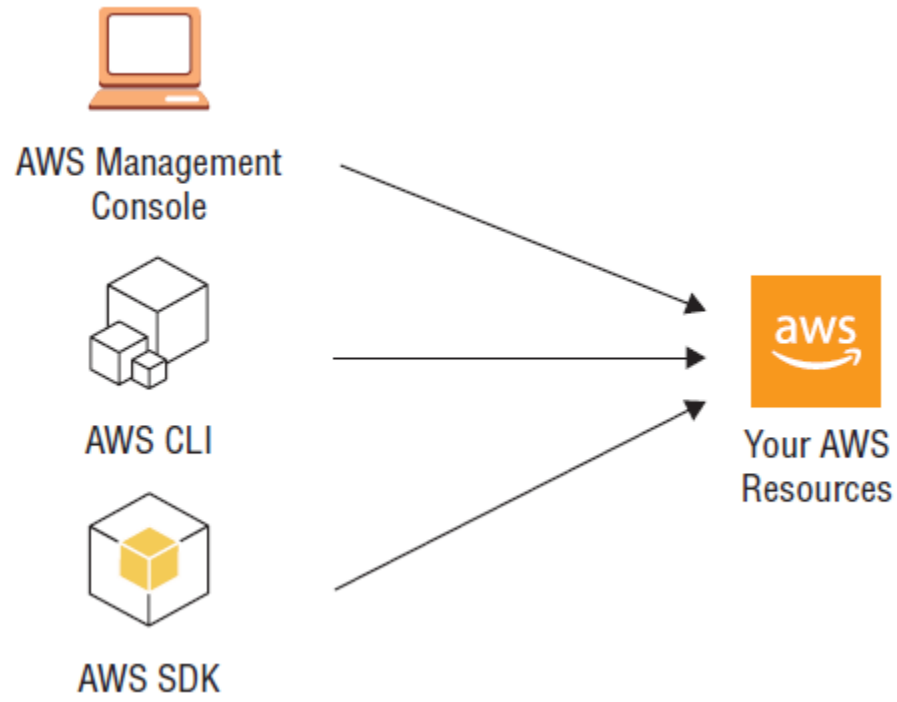


AWS Management Console FrontEnd



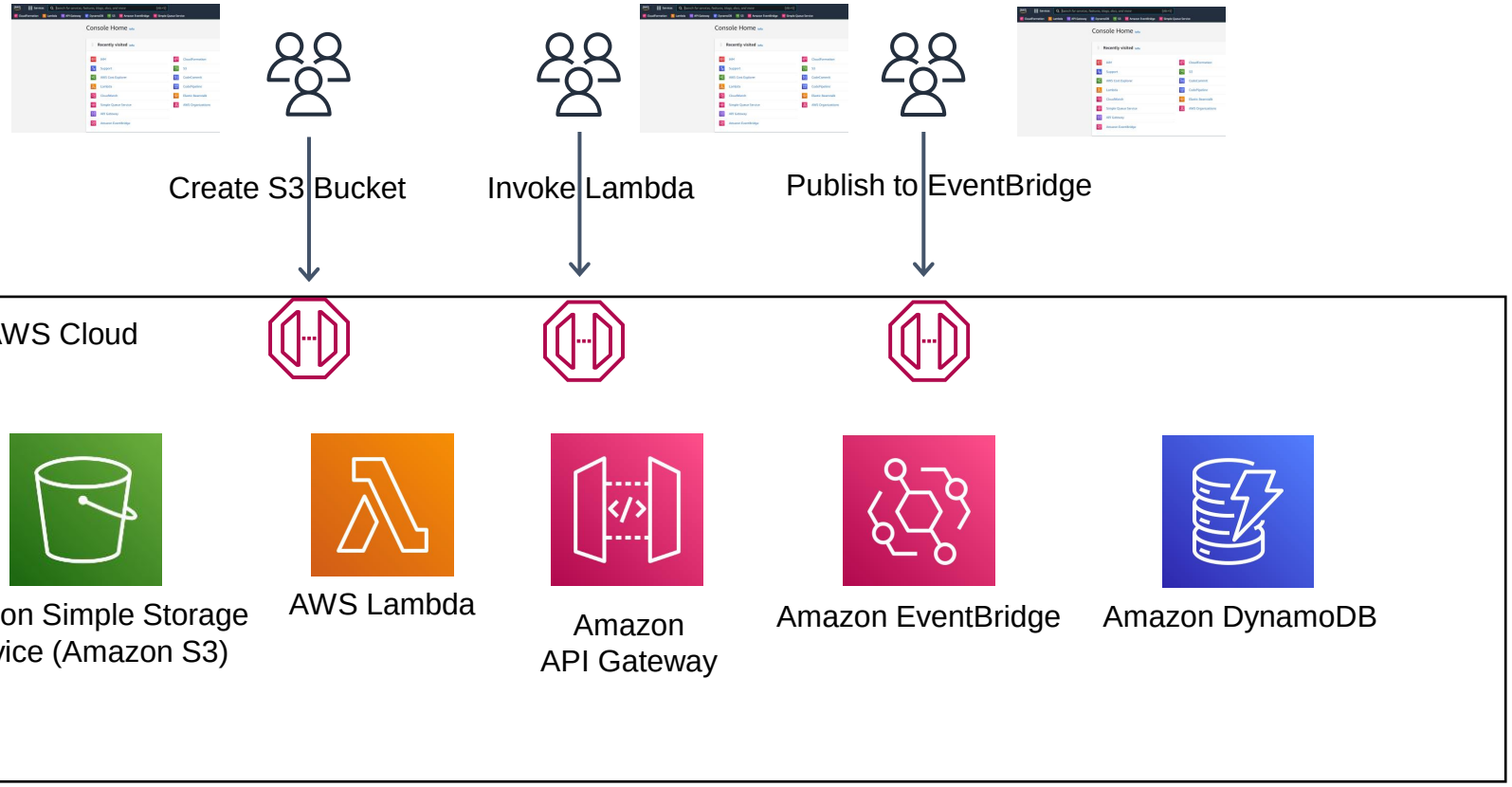
Invoke AWS APIs with Different Ways

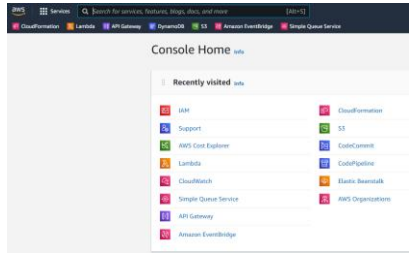
- **AWS Management Console Access**
  You can think as a web application allows us to manage AWS resources for particular AWS accounts.
- **Programmatic Access**
  Gives us to manage AWS resources from our development environments and manage by writing codes.
  - AWS CLI
  - AWS SDK
  - AWS Cloud Formation - IaC
    - AWS SAM
    - AWS CDK



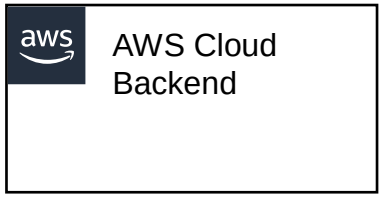
Invoke AWS APIs with Different Ways; AWS Management Console

- **AWS Management Console Access**
  You can think as a web application allows us to manage AWS resources for particular AWS accounts.





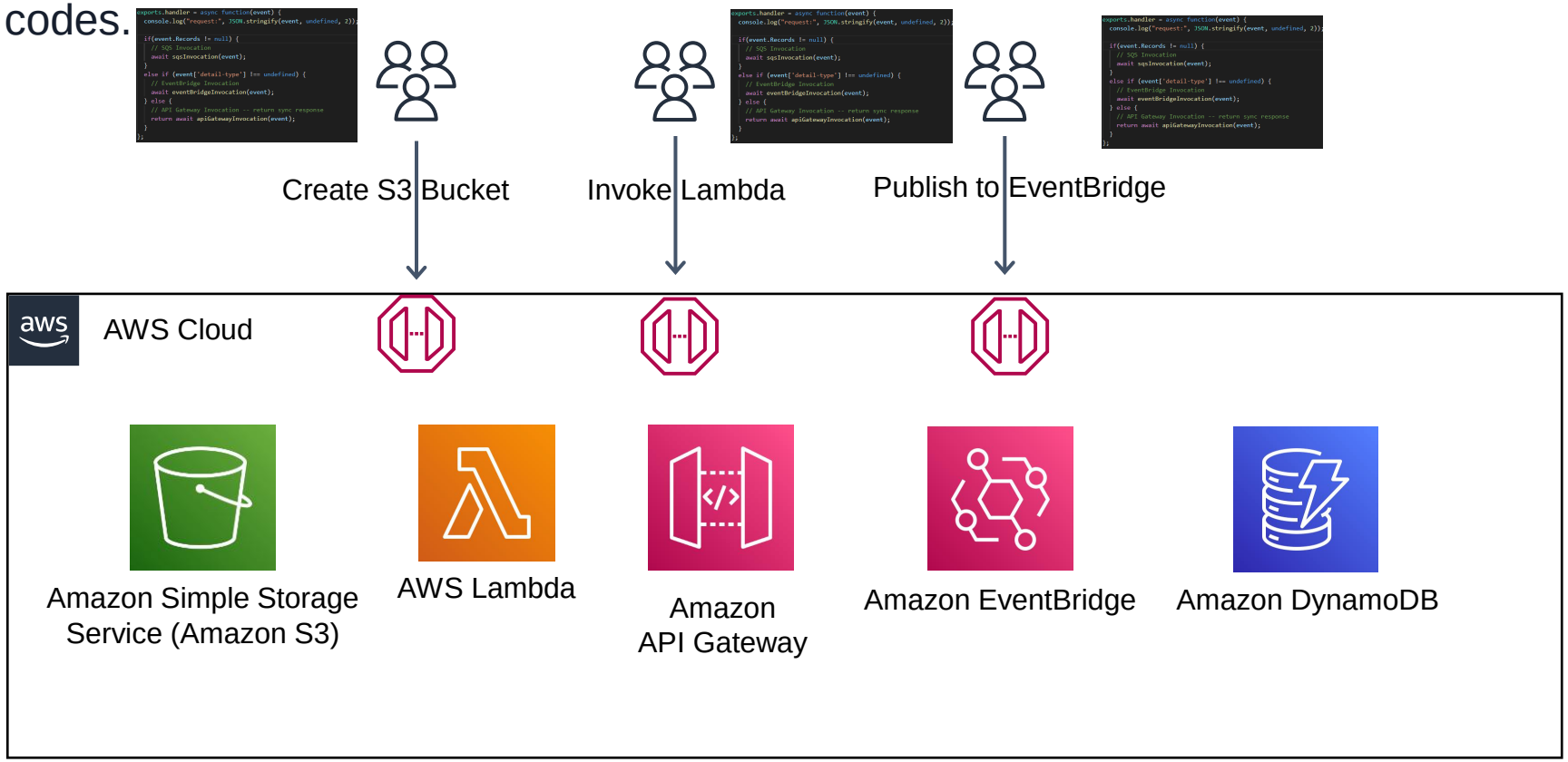
AWS Management Console FrontEnd



Invoke AWS APIs with Different Ways; Programmatic Access

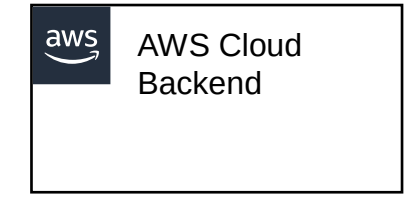
- **Programmatic Access**
  Gives us to manage AWS resources from our development environments and manage

by writing codes.



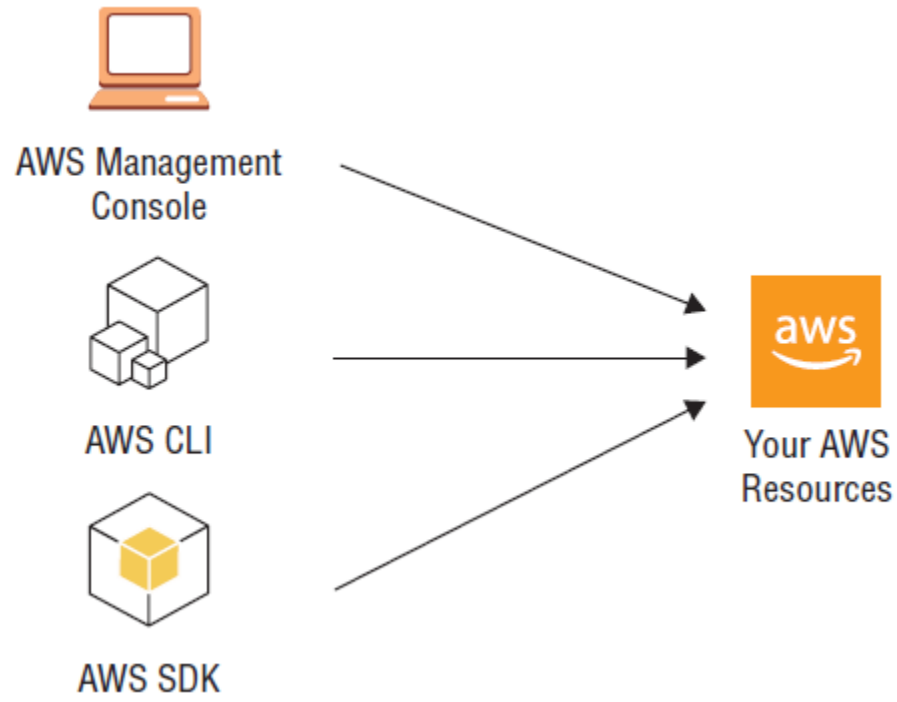

```
exports.handler = async function(event) {
  console.log("request:", JSON.stringify(event, undefined, 2));

  if(event.Records != null) {
    // SQS Invocation
    await sqsInvocation(event);
  }
  else if (event['detail-type'] !== undefined) {
    // EventBridge Invocation
    await eventBridgeInvocation(event);
  } else {
    // API Gateway Invocation -- return sync response
    return await apiGatewayInvocation(event);
  }
};
```

Programmatic Access



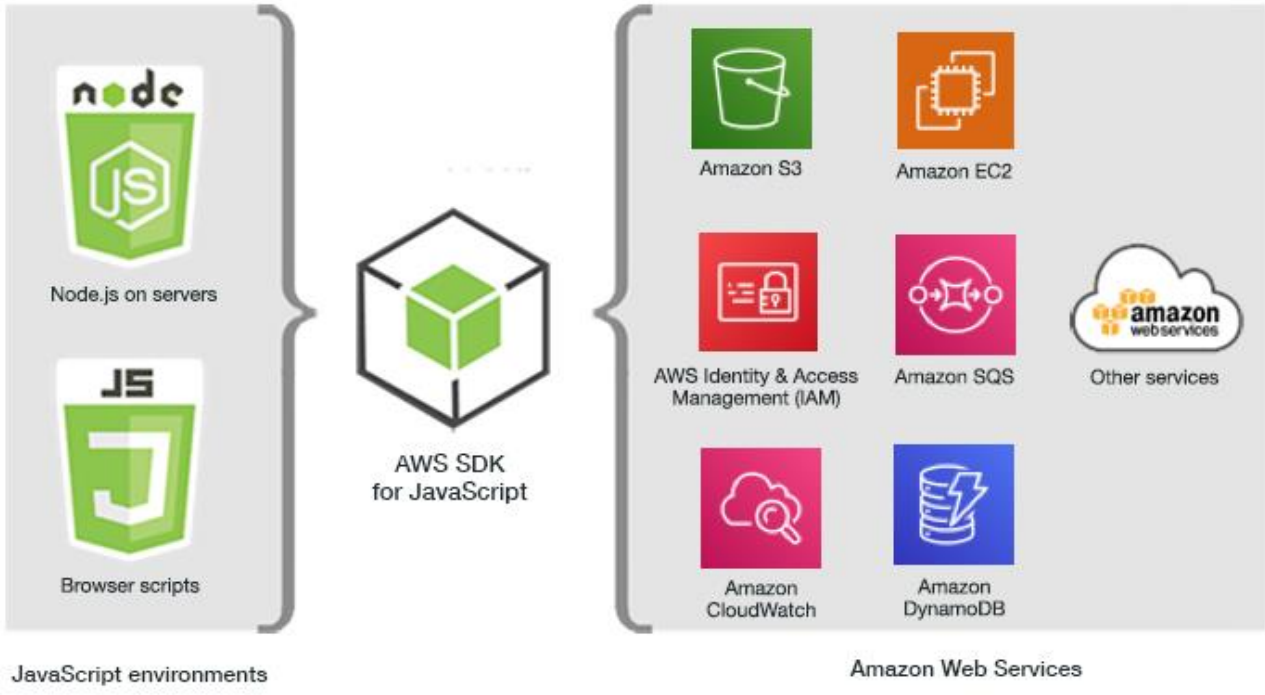
Programmatic Access

- **Programmatic Access**
  Gives us to manage AWS resources from our development environments and manage by writing codes.
  - AWS CLI
  - AWS SDK
  - AWS Cloud Formation - IaC
    - AWS SAM
    - AWS CDK
- **AWS Command Line Interface (CLI)**
  Unified tool to manage your AWS services.
- Control multiple AWS services from the command line and automate them through scripts.
- $ aws ec2 describe-instances
- $ aws ec2 start-instances --instance-ids i-1348636c



Programmatic Access - AWS SDK

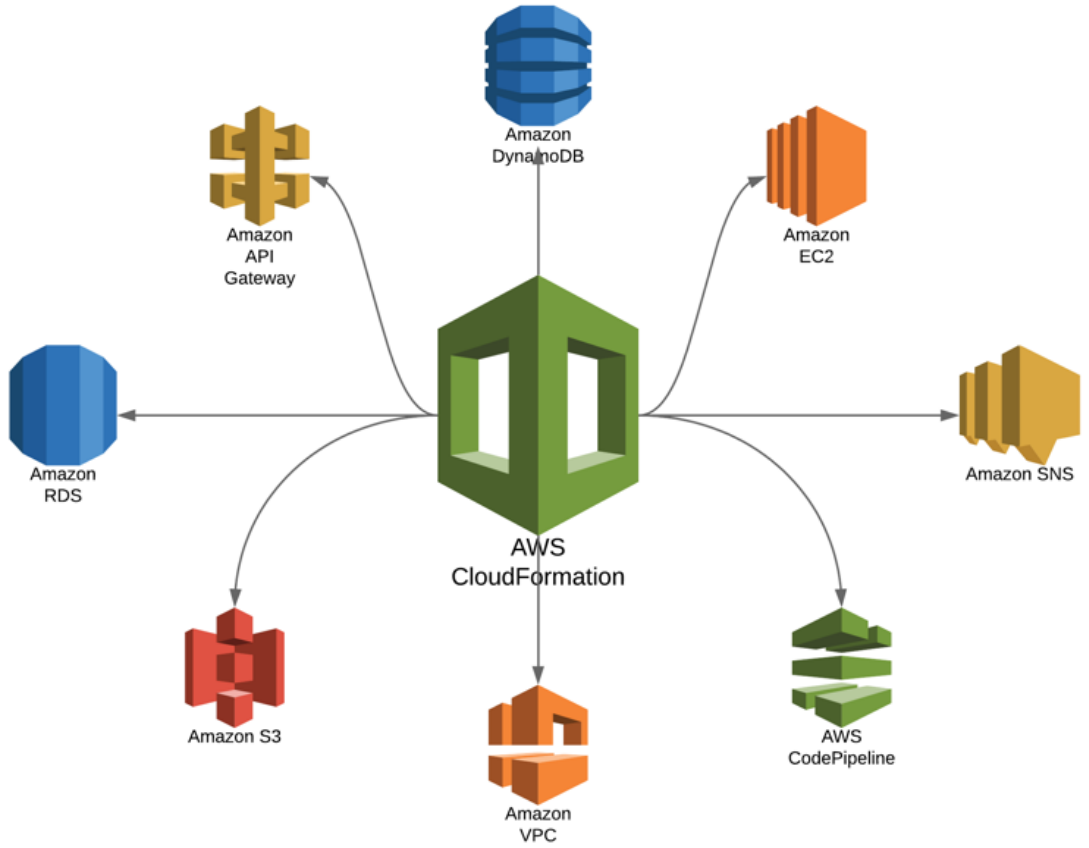
- **AWS SDK - Software Development Kit**
  Simplifies use of AWS Services by providing a set of libraries that are consistent and familiar for developers.
- Several programming languages AWS SDK packages that you can use, like Java, NodeJS, Javascript, .Net, Go and so on.
- Tools to Build on AWS
  Tools for developing and managing applications on AWS
- Most common use cases, perform crud operations on DynamoDB table in your application code with using AWS SDK libraries
- Microservices codes when interacting with AWS DDynamoDB, EventBridge and SQS.



https://aws.amazon.com/blogs/developer/modular-aws-sdk-for-javascript-release-candidate/

Programmatic Access - AWS CloudFormation and AWS CDK

- **AWS CloudFormation**
  Infrastructure as code (IaC) service that allows you to easily model, provision, and manage AWS resources.
  - AWS SAM
  - AWS CDK
- **AWS CDK - Cloud Development Kit**
  Open-source software development framework to define your cloud application resources using familiar programming languages.
- Provisioning cloud infrastructure with using Java, Typescript, Javascript, .Net, go and so on.
- During the course we will install and use all of these access types.



https://aws.amazon.com/cloudformation/

Developing Our First Lambda Function

→ Create and Developing Our First Lambda Function.

AWS Lambda Overview

- The most popular **serverless compute platform** that is using millions of customer
- Running billions of invocations all over the world
- What is AWS Lambda
  - Overview
  - Core Concepts
  - Example Use Cases
  - Main Features
  - Best Practices
  - Walktrough AWS Console – Create Function Run and Test
- **What is AWS Lambda**
  Compute service that **runs code without thinking any servers** or underlying services
- **Serverless function** that you only responsible for your actual code.

AWS Lambda

What is AWS Lambda ?

- AWS Lambda is an **event-driven, serverless computing platform** provided by Amazon as a part of Amazon Web Services. It is a computing service that **runs code** in **response to events** and automatically manages the **computing resources** required by that code. It was introduced in November 2014.[1] – WikiPedia
- Provide to **create functions**, written supported languages and runtimes, and **upload code** to AWS Lambda, and **executes** functions without thinking **scalability** and **availability** issues.
- Don’t need to worry about which AWS resources to launch. Just **Upload** and **Execute** !
- **Supported Runtimes**; Node.js, Python, Java, Go, Ruby, .NET and so on.
- Designed for **event-driven architecture**, so **examples use cases** such as image uploads to Amazon S3, updates to DynamoDB tables, responding to website clicks, and so on.

AWS Lambda

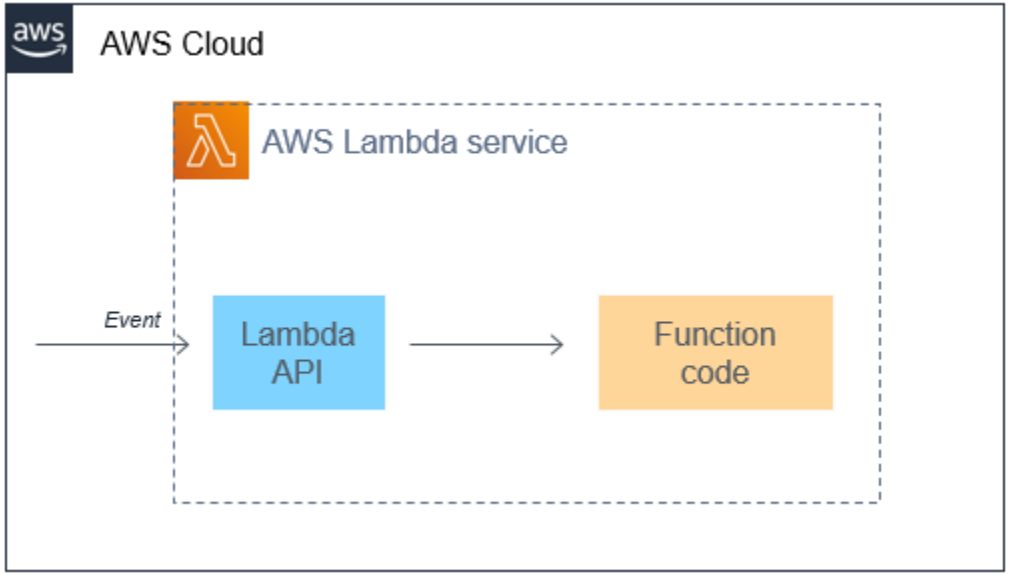
What is AWS Lambda ? - Summarized

- Serverless, **Event-driven** compute service
- Trigger Lambda from over **200 AWS** services
- Run code without provisioning or managing infrastructure. Simply **write** and **upload** code as a .zip file or container image.
- Code execution requests at any **scale**
- **Pay-as-you-go**; Save costs by paying only for the compute time you use—by per-millisecond—instead of provisioning infrastructure
- Optimize **code execution** time and **performance** with the right function memory size.
- Respond to **high demand** in double-digit milliseconds with Provisioned Concurrency.

AWS Lambda

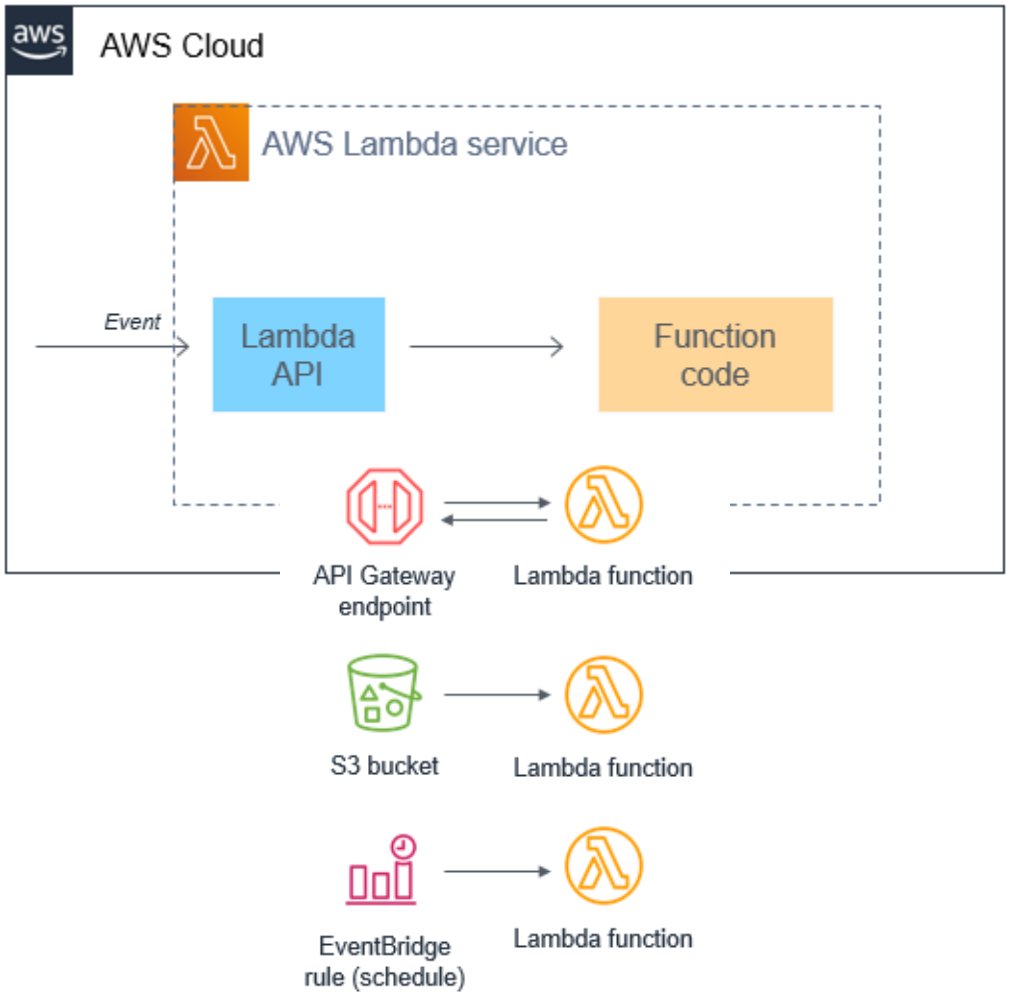
How does AWS Lambda work?

- Each Lambda function runs in its **own container**. You can think every lambda function as a **standalone docker containers**.
- When a function is created, Lambda **packages** it into a new **container** and then **executes** that **container** on a multi-region cloud clusters of servers managed by AWS.
- Each function's container is **allocated** its **necessary RAM** and **CPU** capacity that are **configurable** in AWS Lambda.
- **Charged** based on the **allocated memory** and the amount of **execution time** the function finished.
- AWS Lambda's entire infrastructure layer is **managed by AWS**.
- There is no infrastructure to maintain, you can spend more time on **application code** and your **actual business logics**.



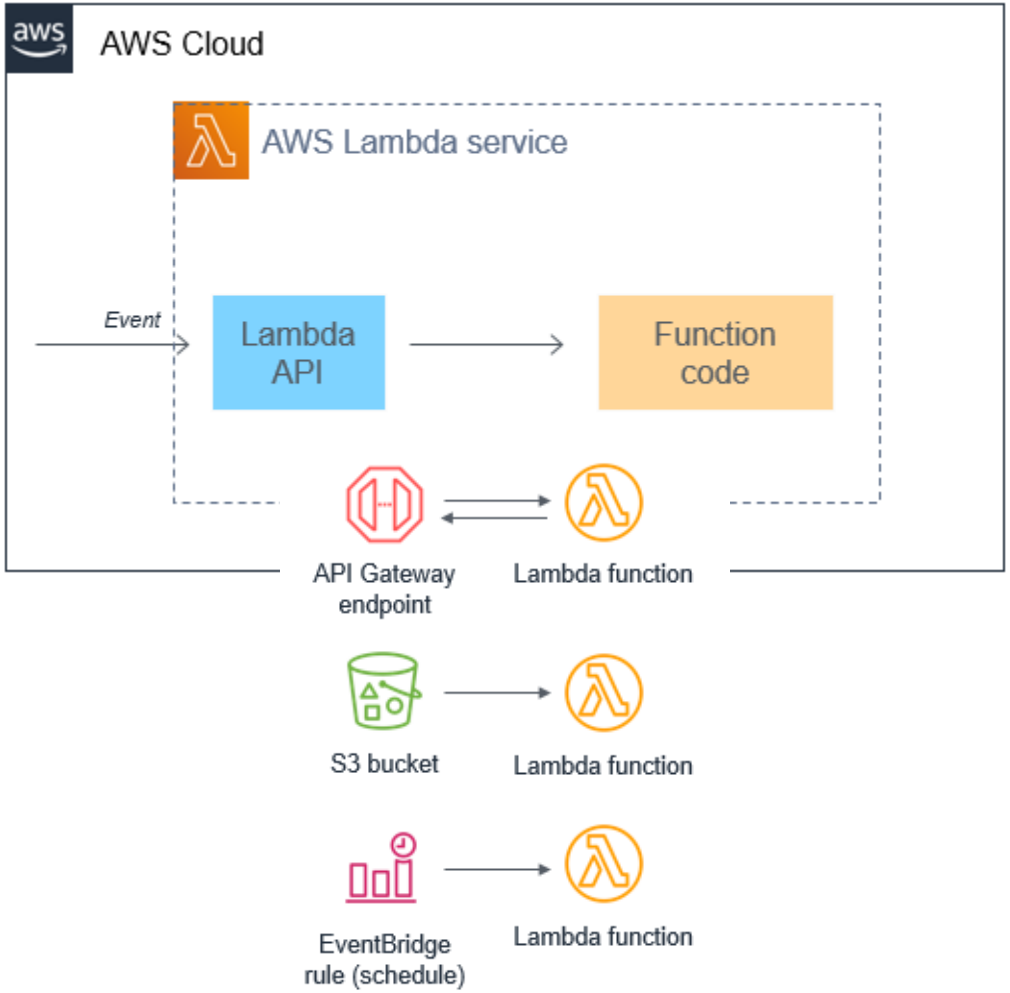
AWS Lambda Main Features

- **Cost Saving with Pay-as-you-go model**
  Customers charged based on the allocated memory and the amount of execution time the function finished.
- **Event-driven Architecture with Lambda**
  Lambda is an on-demand compute service that runs custom code in response to events.
- **Scalability and Availability**
  Lambda can instantly scale up to a large number of concurrent executions
- **Supports Multiple Languages and Frameworks**
  Lambda has native support for a number of programming languages including Java, Go, PowerShell, Node.js, C#, Python, and Ruby code.



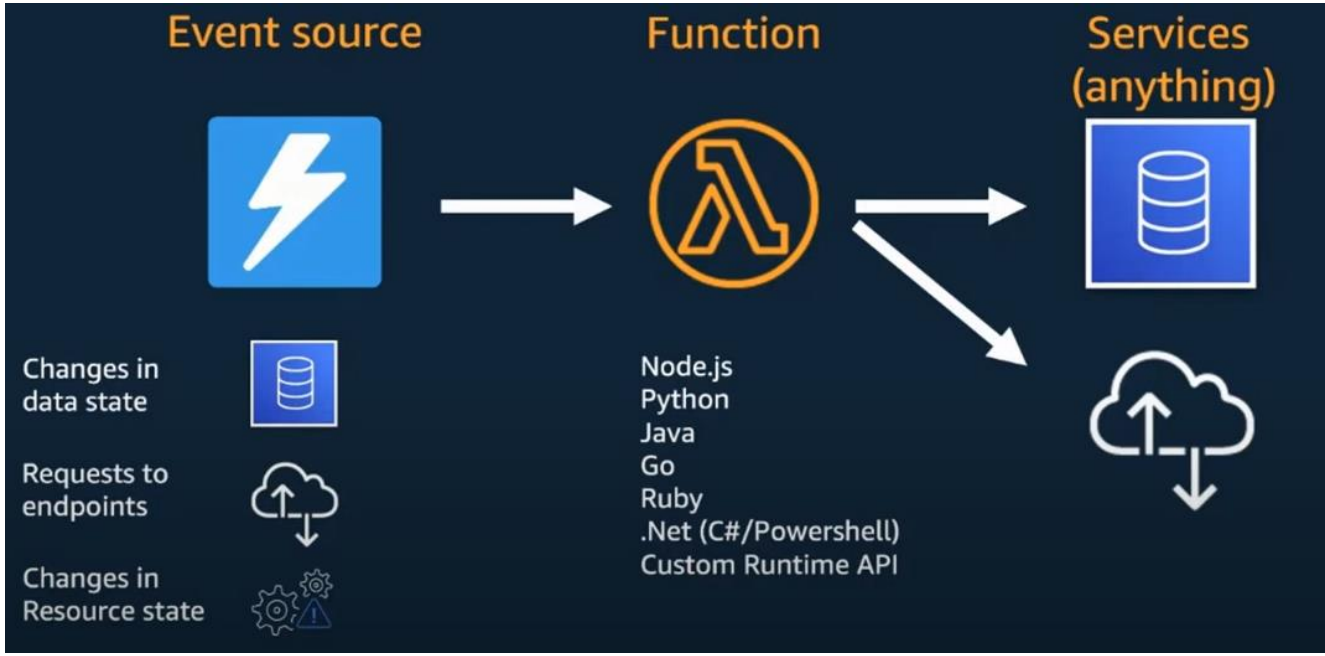
AWS Lambda Key Features

- There are several key features help you develop Lambda applications that are scalable, secure, and easily extensible
- Concurrency and scaling controls
- Functions defined as container images
- Code signing
- Lambda extensions
- Function plans
- Database access
- File systems access



Lambda Event Sources and Destination Trigger Services

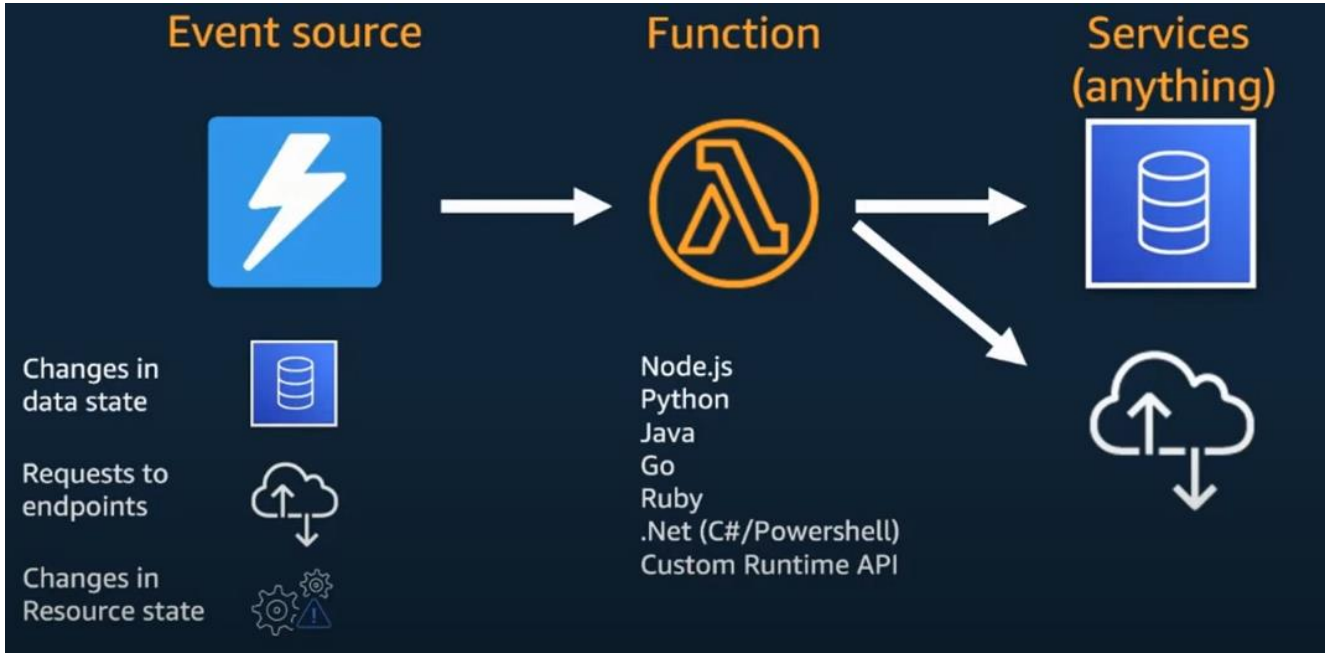
- AWS Lambda integrates with other AWS services to invoke functions or take other actions.
- There is an event source or trigger, and actual Function code and then there is the destination.
- Event source has a number of services; they can be a http call, cron job, uploading an object into S3 bucket, third party call like payment done through stripe
- Triggering event to lambda function, lambda launch the execution environment with different language and runtimes
- Lambda has destinations that can be interaction with your function code



https://www.youtube.com/watch?v=x1Yaxo5uPLM

Use Cases Lambda Event Sources and Destination Trigger Services

- Invoke a function in response to resource lifecycle events, such as with Amazon Simple Storage Service (Amazon S3)
- Respond to incoming HTTP requests. Using Lambda with API Gateway.
- Consume events from a queue. Using Lambda with Amazon SQS. Lambda poll queue records from Amazon SQS.
- Run a function on a schedule. Using AWS Lambda with Amazon EventBridge (CloudWatch Events).



https://www.youtube.com/watch?v=x1Yaxo5uPLM

List of Services Lambda Event Sources

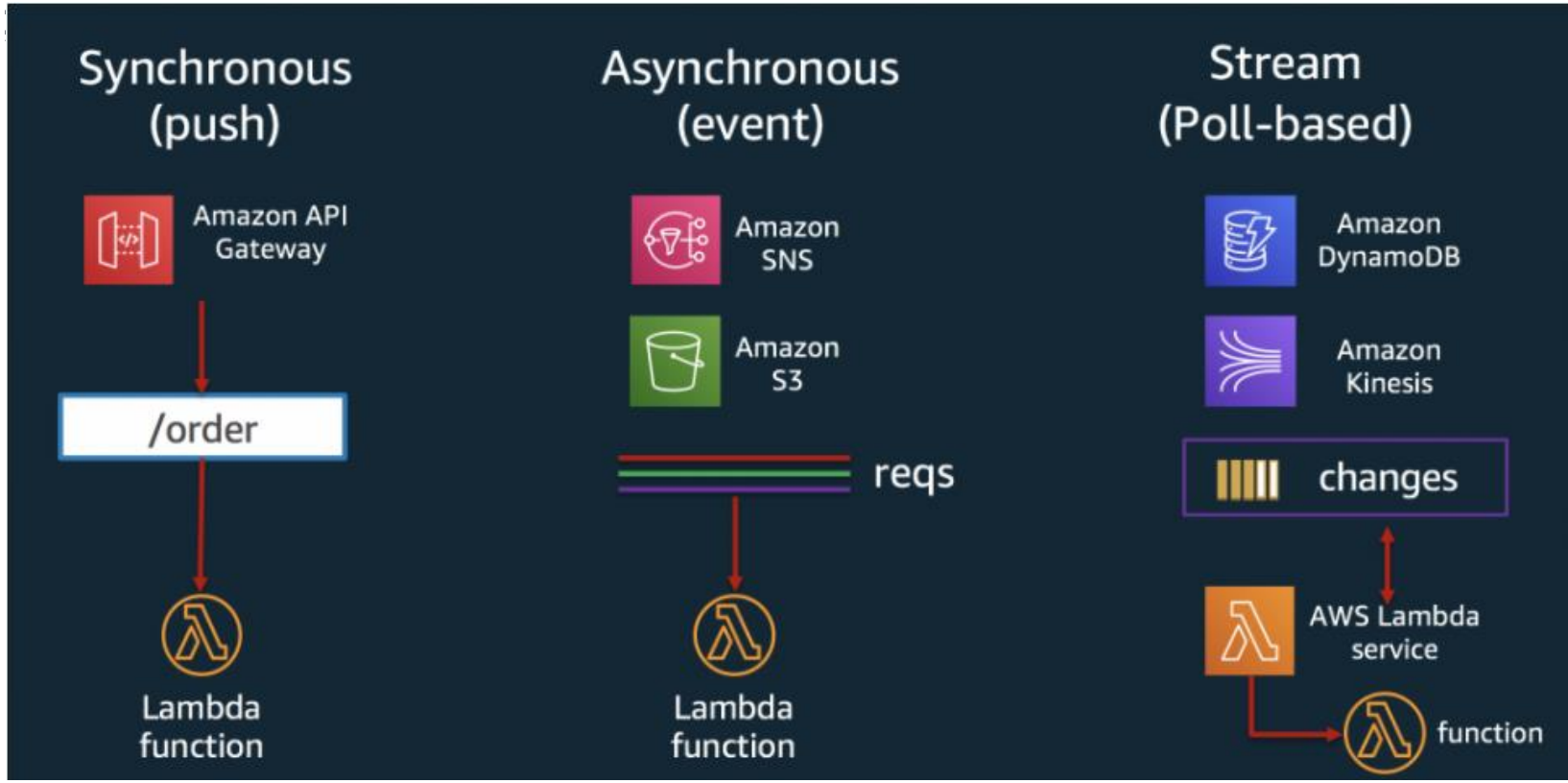
Service	Method of invocation
Amazon API Gateway	Event-driven; synchronous invocation
AWS CloudFormation	Event-driven; asynchronous invocation
Amazon CloudFront (Lambda@Edge)	Event-driven; synchronous invocation
Amazon EventBridge (CloudWatch Events)	Event-driven; asynchronous invocation
Amazon CloudWatch Logs	Event-driven; asynchronous invocation
AWS CodeCommit	Event-driven; asynchronous invocation
AWS CodePipeline	Event-driven; asynchronous invocation
Amazon Cognito	Event-driven; synchronous invocation
AWS Config	Event-driven; asynchronous invocation
Amazon Connect	Event-driven; synchronous invocation
Amazon DynamoDB	Lambda polling
Amazon Elastic File System	Special integration

https://docs.aws.amazon.com/lambda/latest/dg/lambda-services.html#eventsources-sqs

AWS Lambda Invocation Types

- Triggered lambda functions with different AWS Lambda Invocation Types
- AWS Lambda has 3 Invocation Types;
- **Lambda Synchronous invocation**
- **Lambda Asynchronous invocation**
- **Lambda Event Source Mapping**

with polling invocation



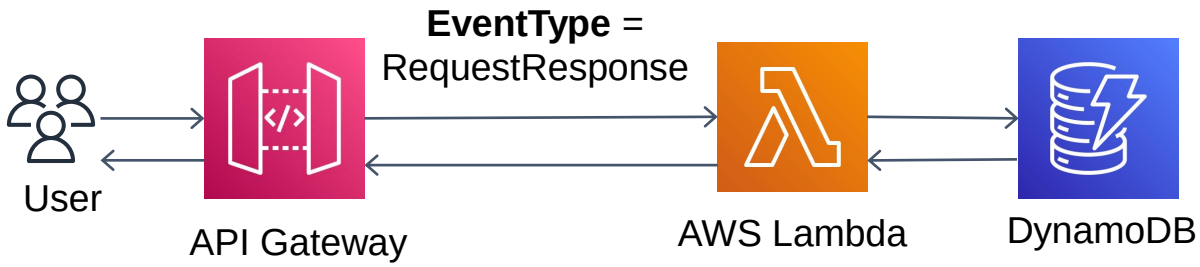
https://aws.amazon.com/blogs/architecture/understanding-the-different-ways-to-invoke-lambda-functions/

AWS Lambda Synchronous Invocation

- **Execute immediately** when you perform the Lambda Invoke API call.
- **Wait** for the function to process the function and return back to response.
- **API Gateway + Lambda + DynamoDB**
- **Invocation-type** flag should be **“RequestResponse”**
- Responsible for **inspecting** the **response**

  and **determining** if there was an **error**

  and **decide** to **retry** the invocation



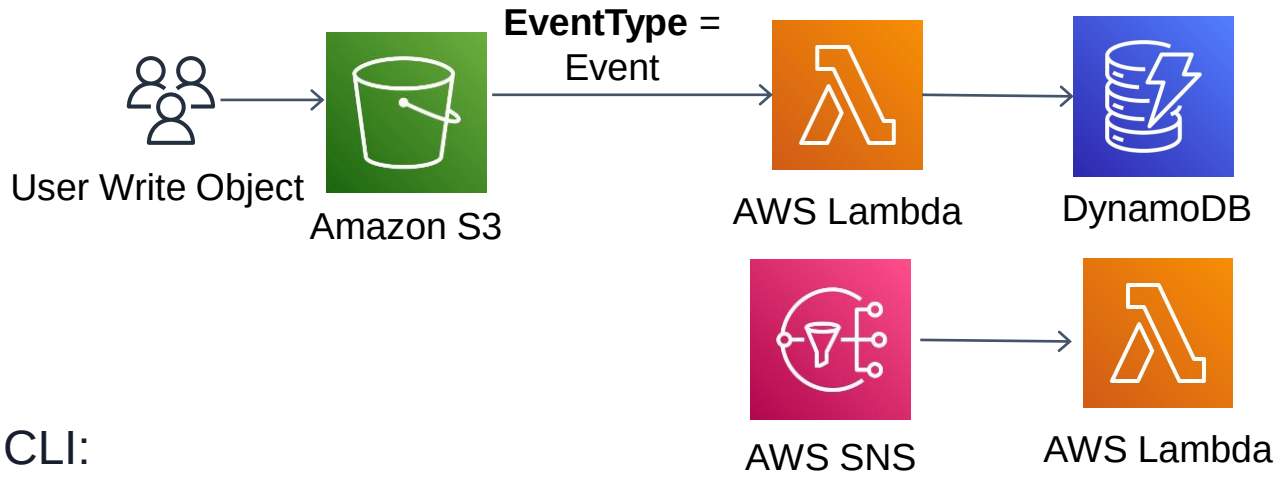
- Example of synchronous invocation using the AWS CLI:
  aws lambda invoke —function-name MyLambdaFunction **—invocation-type RequestResponse** —payload '{ "key": "value" }’
- Triggered **AWS services** of **synchronous invocation**; ELB (Application Load Balancer), Cognito, Lex, Alexa, API Gateway, CloudFront, Kinesis Data Firehose

AWS Lambda Asynchronous Invocation

- Lambda **sends** the **event** to a **internal queue** and returns a **success response** without any additional information
- Separate process **reads events** from the **queue** and **runs** our lambda function
- **S3 / SNS + Lambda + DynamoDB**
- **Invocation-type** flag should be **“Event”**
- AWS Lambda sets a **retry policy**

  Retry Count = 2

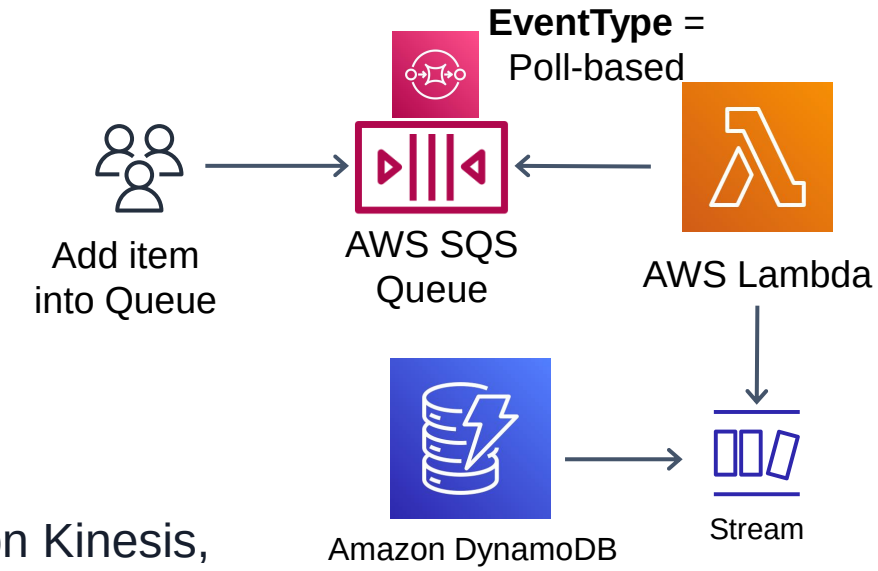
  Attach a Dead-Letter Queue (DLQ)



- Example of asynchronous invocation using the AWS CLI:
  aws lambda invoke —function-name MyLambdaFunction **—invocation-type Event** —payload '{ "key": "value" }'
- Triggered **AWS services** of **asynchronous invocation**; S3, EventBridge, SNS, SES, CloudFormation, CloudWatch Logs, CloudWatch Events, CodeCommit

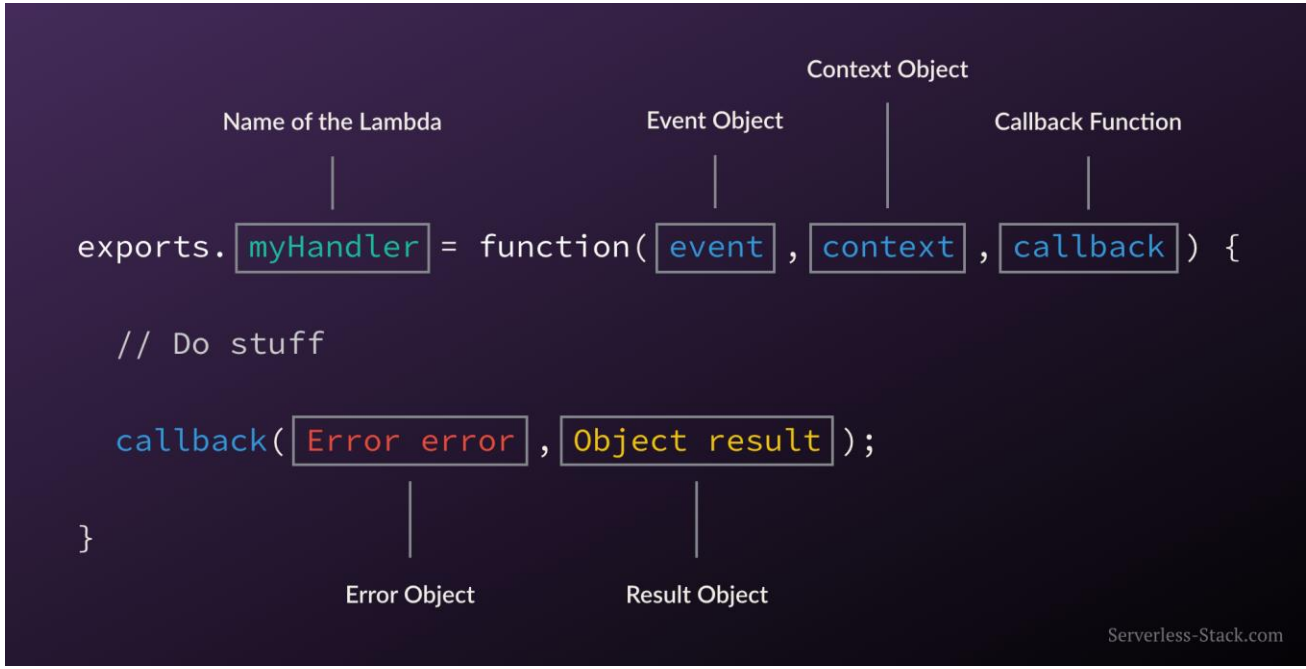
AWS Lambda Event Source Mapping with Polling Invocation

- **Pool-Based invocation** model allows us to **integrate** with **AWS Stream** and **Queue based services**.
- Lambda will **poll** from the AWS SQS or Kinesis streams, retrieve records, and invoke functions.
- Data stream or queue are **read** in **batches**,
- The function receives multiple items when execute function.
- **Batch sizes** can configure according to service types
- **SQS + Lambda**
- **Stream based processing** with **DynamoDB Streams + Lambda**
- Triggered **AWS services** of **Event Source Mapping invocation**; Amazon Kinesis, DynamoDB, Simple Queue Service (SQS)



Lambda Function Code

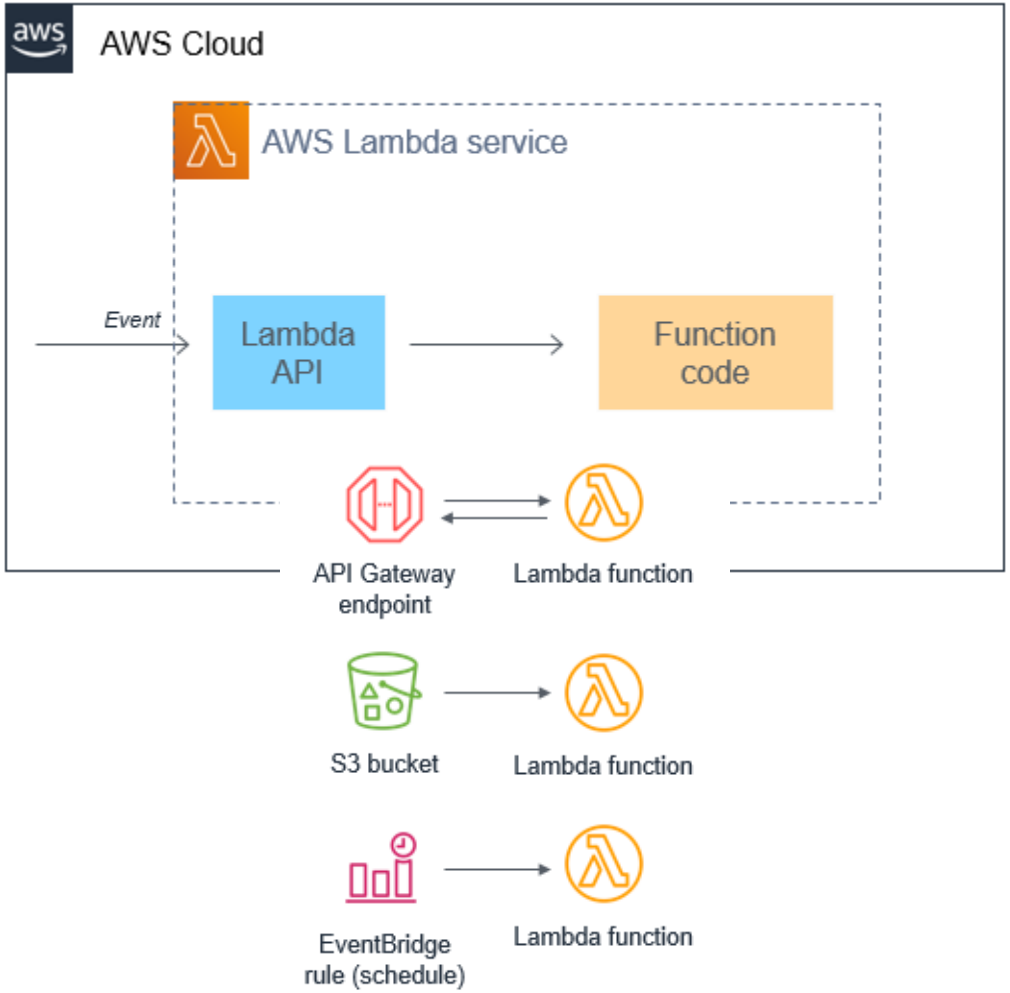
- Lambda runs instances of your function to process events. You can invoke your function directly using the Lambda API, or you can configure an AWS service or resource to invoke your function.
- Lambda function has code to process the events that you pass into the function or that other AWS services send to the function with event json object.
- The event object contains all the information about the event that triggered this Lambda.
- The context object contains info about the runtime our Lambda function
- Return the callback function with the results



https://serverless-stack.com/chapters/what-is-aws-lambda.html

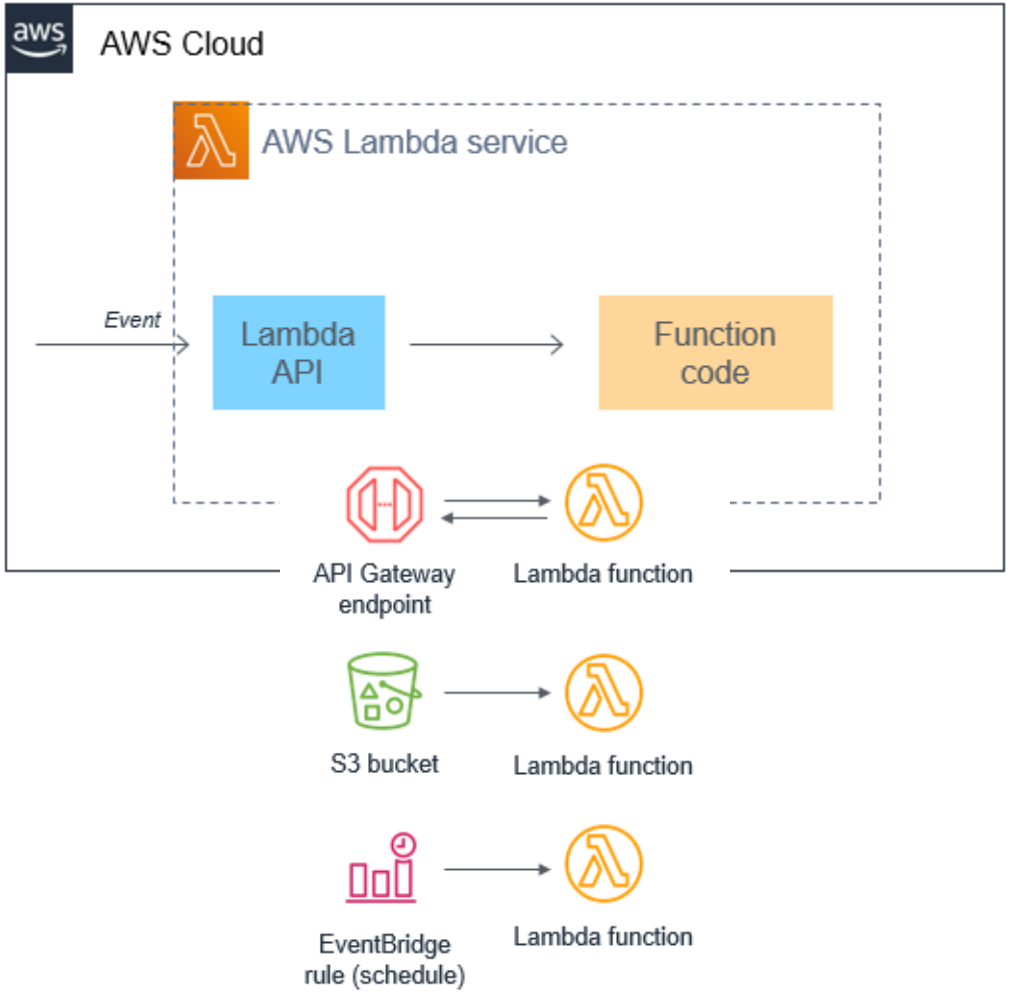
Key concepts of Lambda Function Code

- **Runtime:** select the runtime as part of configuring the function, and Lambda loads that runtime when initializing the environment.
- **Handler:** function runs starting at the handler method.
- **Function:** is a resource that you can invoke to run your code in Lambda.
- **Trigger:** is a resource or configuration that invokes a Lambda function.
- **Event:** is a JSON-formatted document that contains data for a Lambda function to process.
- **Execution environment:** provides a secure and isolated runtime environment for your Lambda function.



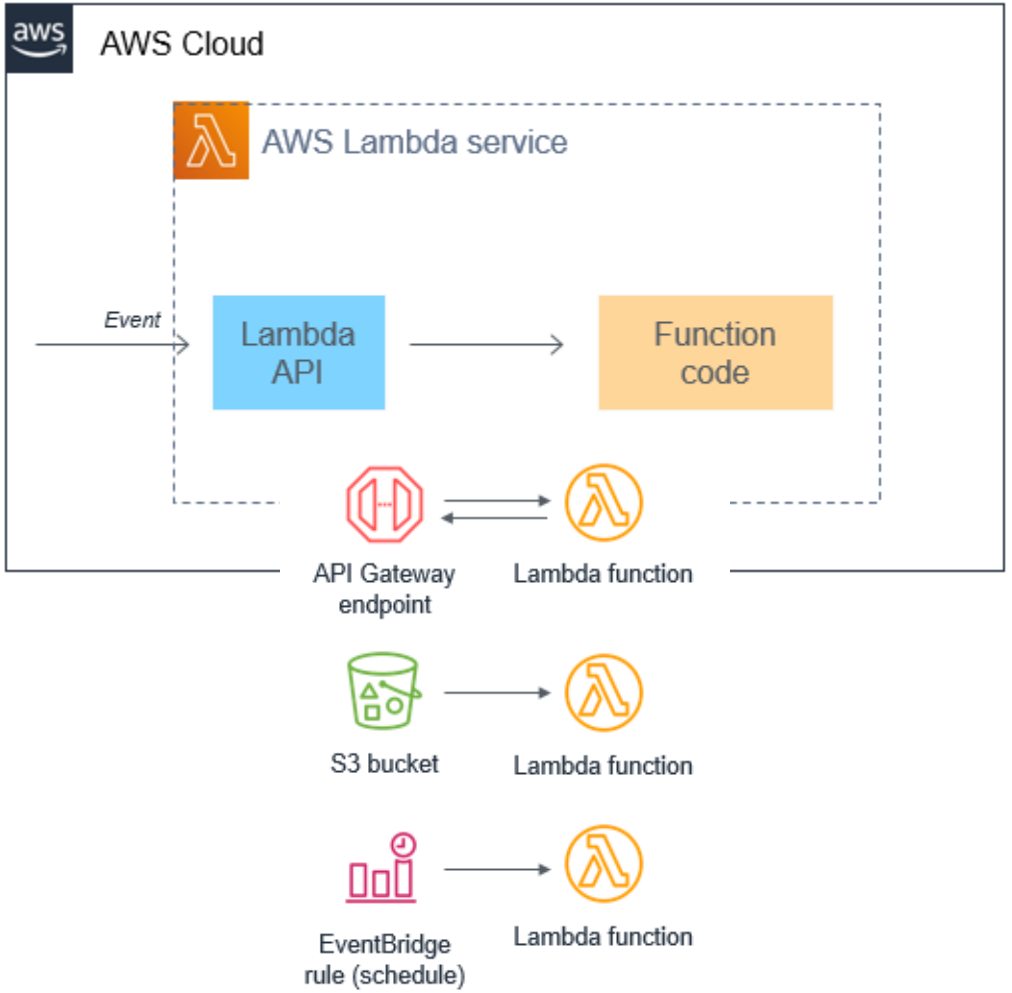
Key concepts of Lambda Function Code

- **Layer:** can contain libraries, a custom runtime, data, or configuration files. Using layers reduces the size of uploaded deployment archives and makes it faster to deploy your code.
- **Concurrency:** is the number of requests that your function is serving at any given time. When your function is invoked, Lambda provisions an instance of it to process the event. When the function code finishes running, it can handle another request.
- **Destination:** is an AWS resource where Lambda can send events from an asynchronous invocation. configure a destination for events that fail processing like setting DLQ for Lambda fails.



Best Practices of Lambda Function Code

- Take advantage of **environment reuse**, and check that background processes have completed.
- Manage **database connection pooling** with a database proxy. Persist state data externally.
- Configure Function with **Resource-based policy**. Resource-based policy grants permissions to invoke your lambda function. Execution role defines a function's permission to interact with resources.
- Use **Environment variables** to store secrets securely and adjust your function's behavior without updating code. An environment variable is a pair of strings that are stored in a function's version-specific configuration.

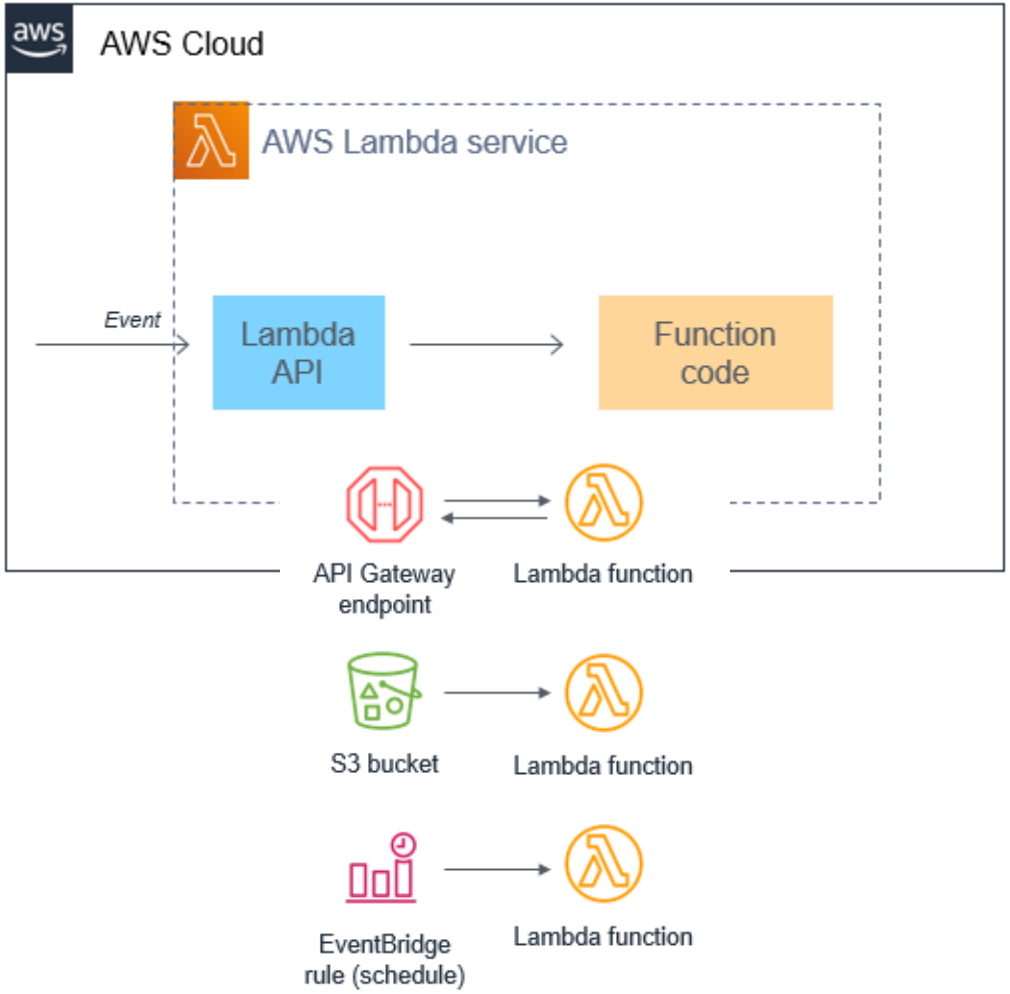


DEMO - Lambda Walkthrough with AWS Console

→ Walkthrough Lambda with AWS Management Console.

AWS Lambda Best Practices and Event-driven Architecture

- AWS Lambda **design principles** and the **best practices** when developing our Lambda-based Event-driven Serverless e-commerce applications.
- Lambda is very **good fit** with **Event-driven Architectures.**
- AWS services generate events for **communicating** each other, most of AWS services are **event sources** for Lambda.
- Lambda always **handle** all **interactions** with the **Lambda API** and there is no direct invocation of functions from outside the service.
- The main purpose of lambda functions is to **handle events.** Even the simplest Lambda-based application uses at least **one event.**
- Lambda functions are limited to **15 minutes** in duration.
- An event triggering a Lambda function could be **almost anything**.



AWS Lambda Events

- The event is a **JSON object** that contains all information about what happened.
  - Represents a change in the system state.
- The first parameter of every **Lambda handler function** contains the event json object.
  - With using this event json object, we can access the event parameters into lambda function.
- An event could be custom-generated from another microservice,
  - New order generated in an ecommerce application.
- The event also can be generated from **existing AWS service**
  - Amazon SQS when a new queue message is available in a queue
- **Event-driven architectures** rely on creating events into all application state changes that are observable by other services
  - Loosely coupled services.

Event name

NewOrderEvent

```
{
  "source": "myApplication",
  "detail": "submitOrder",
  "customerId": "customer123",
  "orderId": "order-A1234B56",
  "paymentStatus": "open",
  "cart": [
    {
      "product12": {
        "qty": 2,
        "itemPx": 35.22,
        "currency": "USD"
      },
      "product44": {
        "qty": 5,
        "itemPx": 71.57,
        "currency": "USD"
      }
    }
  ],
  "timestamp": 1607774286
}
```

index.js

```
const AWS = require('aws-sdk')
AWS.config.region = process.env.AWS_REGION

const s3 = new AWS.S3()

exports.handler = async (event) => {

    console.log(JSON.stringify(event,0, null))

    const Bucket = event.Records[0].s3.bucket.name
    const Key = decodeURIComponent(event.Records[0].s3.object.key.replace(/\+/g, ' '))

    const result = await s3.getObject({
        Bucket,
        Key
    }).promise()

    const data = result.Body
    console.log('PDF text length: ', data.length)

}
```

1

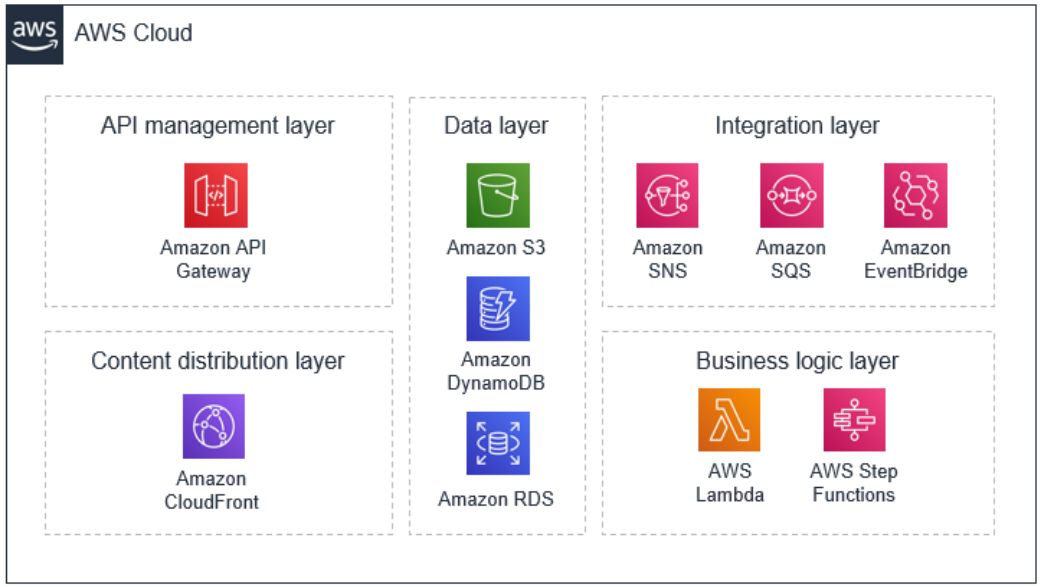
event.source

event.detail

2

AWS Lambda Best Practices and Event-driven Architecture

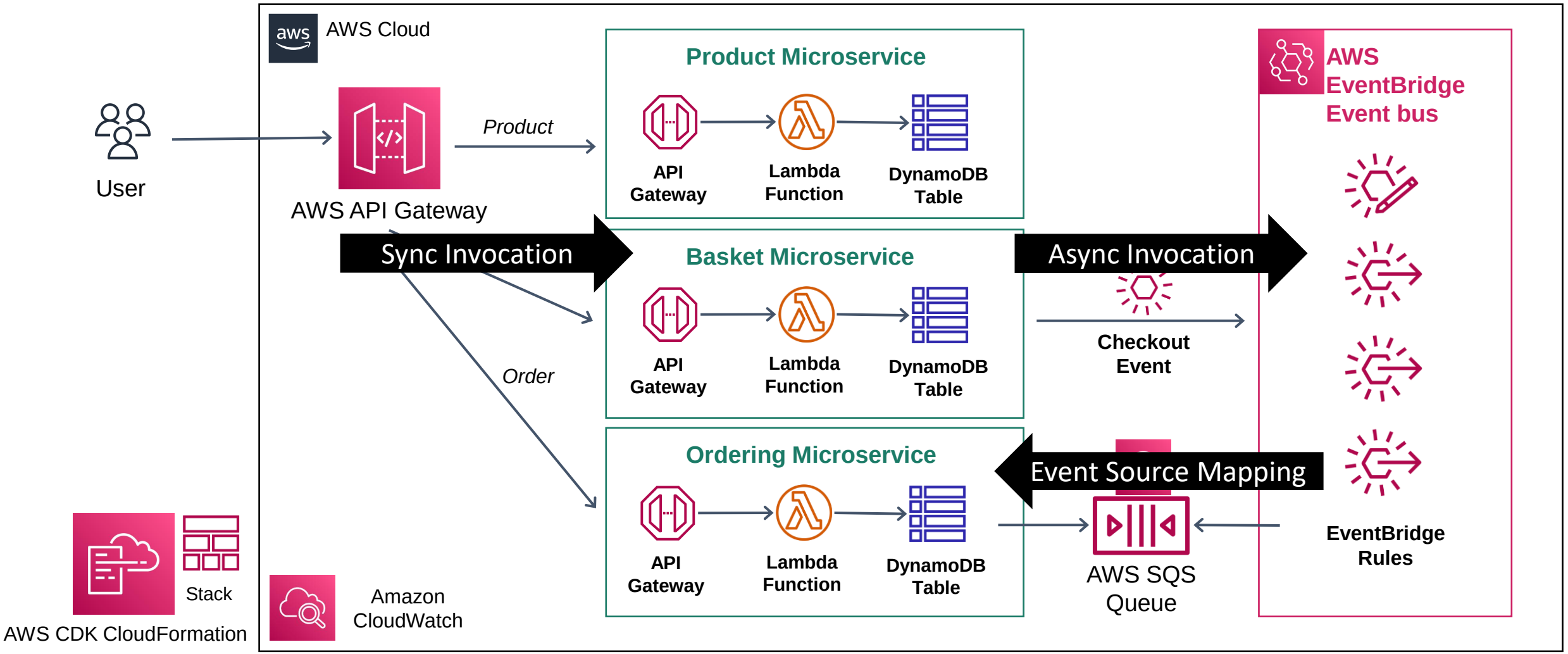
- Most Lambda-based applications use a **combination of AWS services** for different requirements about **Storage**, **API Management** and **integrating** with other system and services.
- Lambda is **connecting between services**, providing business logic to transform data that moves between services.
- You can find mostly **integrated AWS Services** which using Lambda functions.
- **Design patterns** in **Distributed architectures** with AWS Lambda
- When your application needs one of these patterns, we can use the corresponding AWS service.
- These services and patterns are **designed to integrate** with AWS Lambda functions



Pattern	AWS service
Queue	Amazon SQS
Event bus	Amazon EventBridge
Publish/subscribe (fan-out)	Amazon SNS
Orchestration	AWS Step Functions
API	Amazon API Gateway
Event streams	Amazon Kinesis

https://aws.amazon.com/blogs/compute/operating-lambda-design-principles-in-event-driven-architectur

AWS Serverless Microservices with Event-Driven Lambda Calls

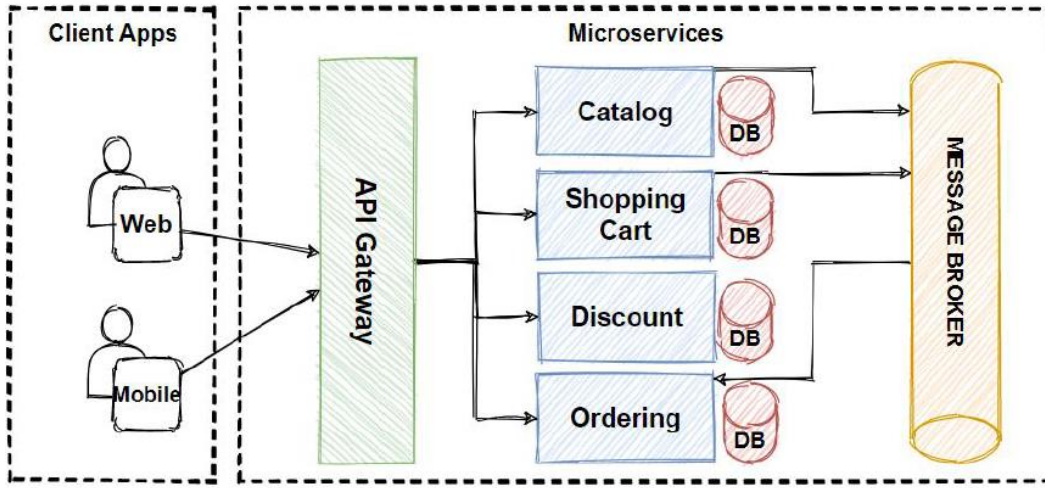


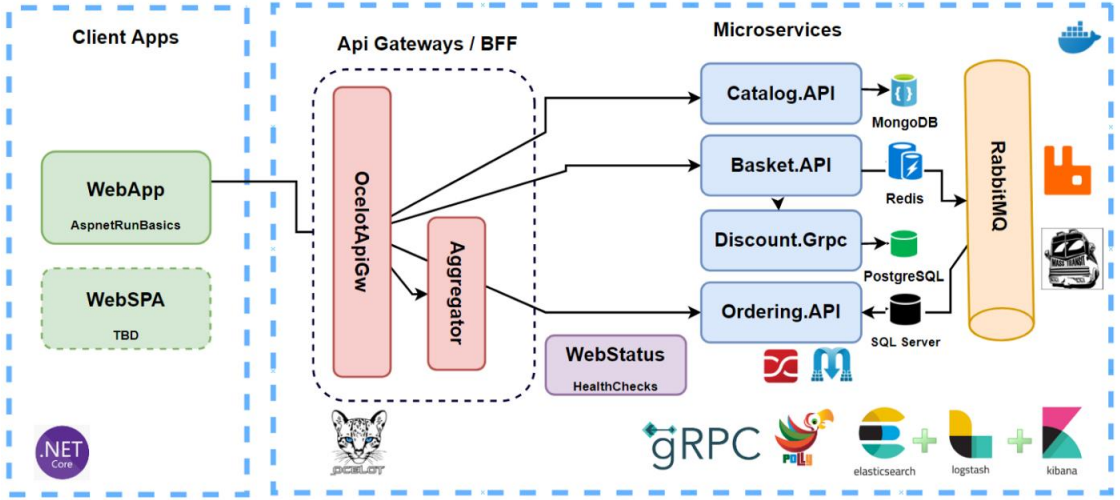
Event-Driven Microservices Architectures

→ Architecture Recap : Event-Driven Microservices Architectures

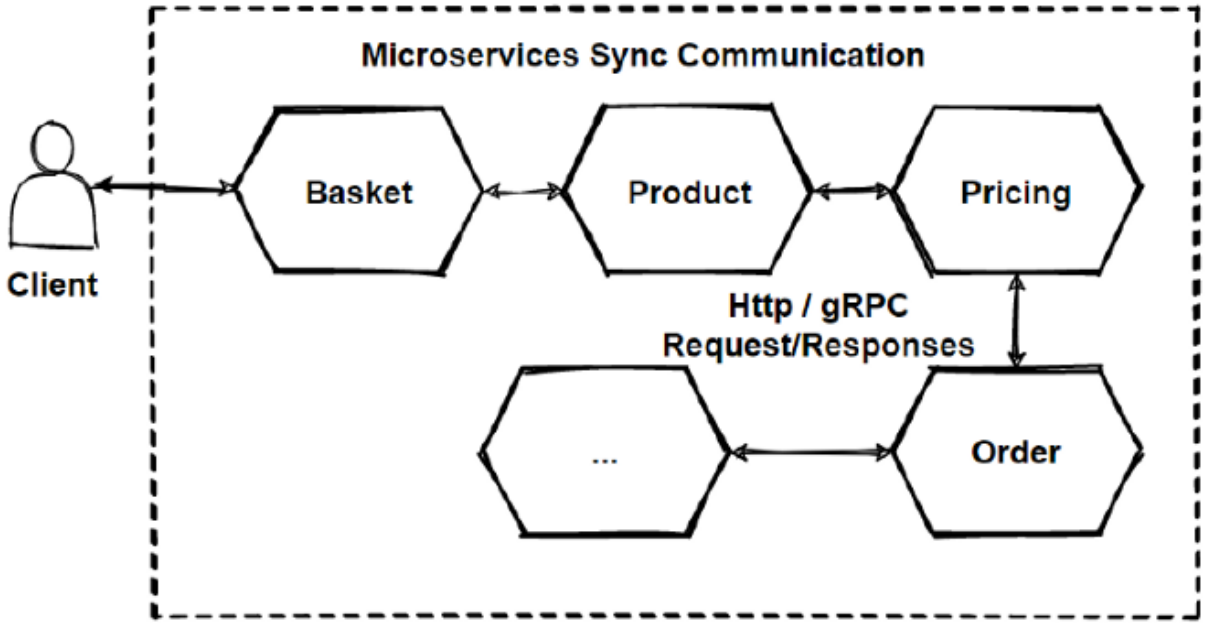
Microservices Architecture

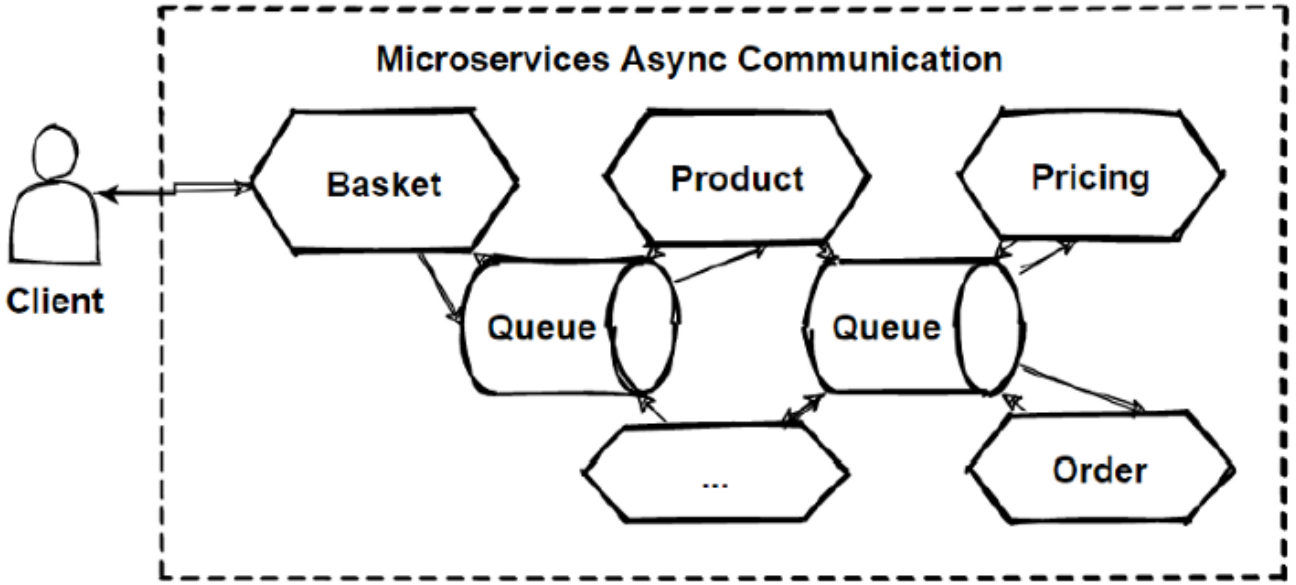
- Microservice are **small business services** that can work together and can be **deployed autonomously** / independently.
- Communicate with each other by talking **over the network**, can be **deployed independently**.
- Minimum of centralized management of these services, it offers the opportunity to work with many **different technologies**.
- Developing a single application as a suite of small services, each running in its **own process** and **communicating** with lightweight mechanisms, often an **HTTP** or **gRPC API.**
- Built around **business capabilities** and **independently deployable** by fully **automated deployment** process.
- Cloud native architectural approach in which applications composed of many **loosely coupled** smaller components.





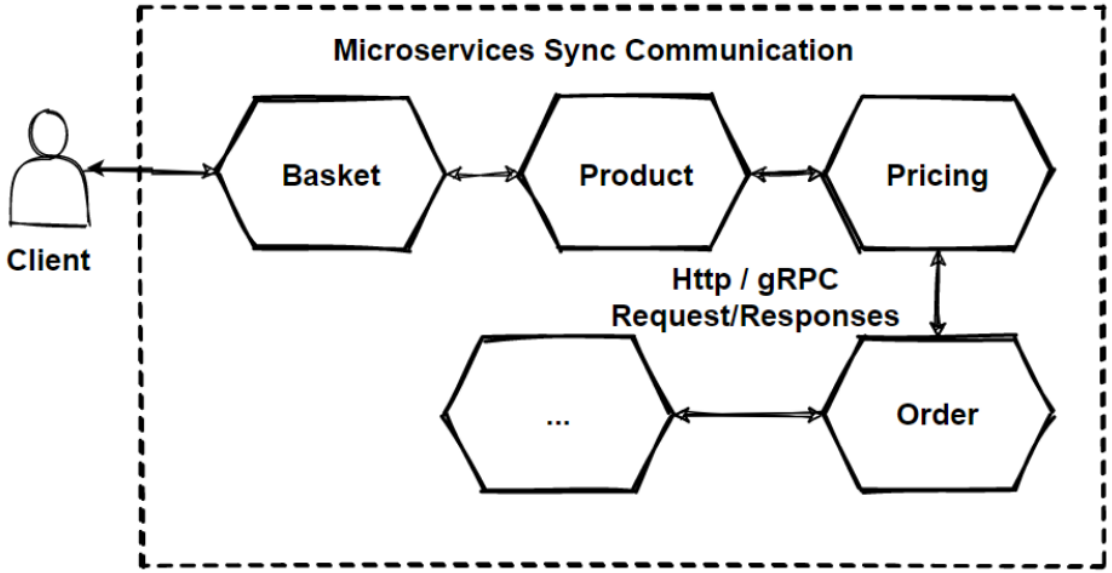
Microservices Communications





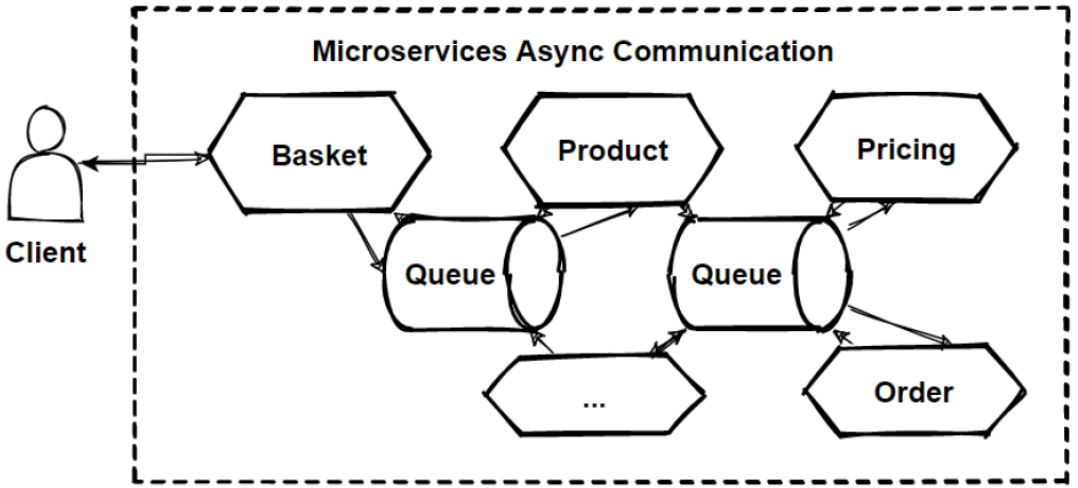
What is Synchronous communication ?

- The client **sends a request** and **waits for a response** from the service.
- Client code **block their thread**, until the response reach from the server.
- **Synchronous communication** protocols can be **HTTP** or **HTTPS**.
- The client sends a request with using **http protocols** and **waits for a response** from the service.
- Client call the server **and block client** their operations.
- Client **code will continue** its task when it **receives** the **HTTP** server **response**.



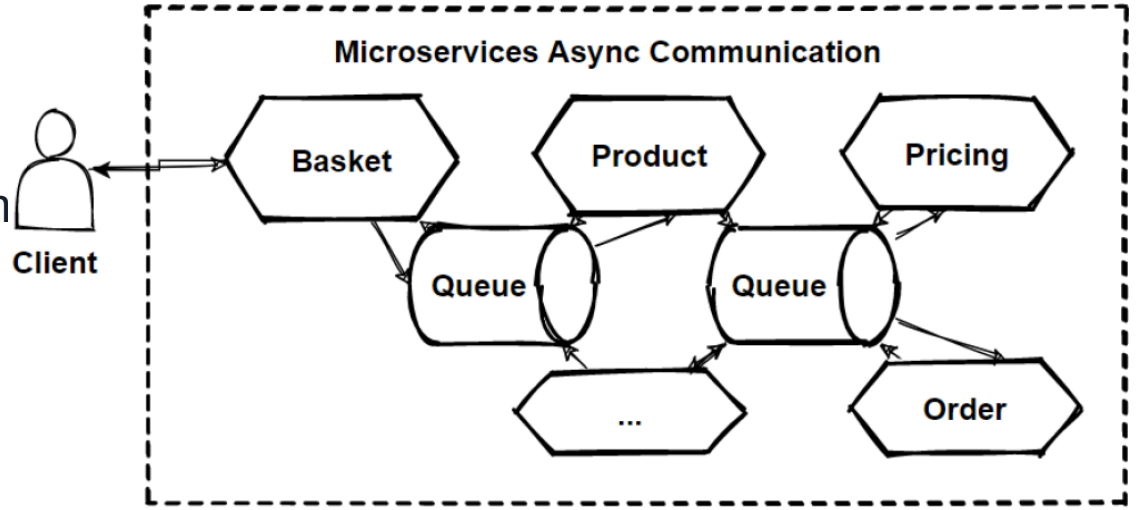
What is Asynchronous communication ?

- The client sends a request but it **doesn't wait for a response** from the service.The client should **not have blocked a thread** while waiting for a response.
- Using **AMQP protocols**, the client sends the message with using message broker systems like **Kafka** and **RabbitMQ** queue.
- The message producer usually **does not wait for a response**. This message **consume** from the **subscriber** systems in **async way**, and no one waiting for response suddenly.
- An **asynchronous** systems can be implemented in a **one-to-one(queue)** mode or **one-to-many (topic)** mode.
- In **a one-to-one(queue)** implementation there is a single producer and single receiver.
- In **one-to-many (topic)** implementation has Multiple receivers.



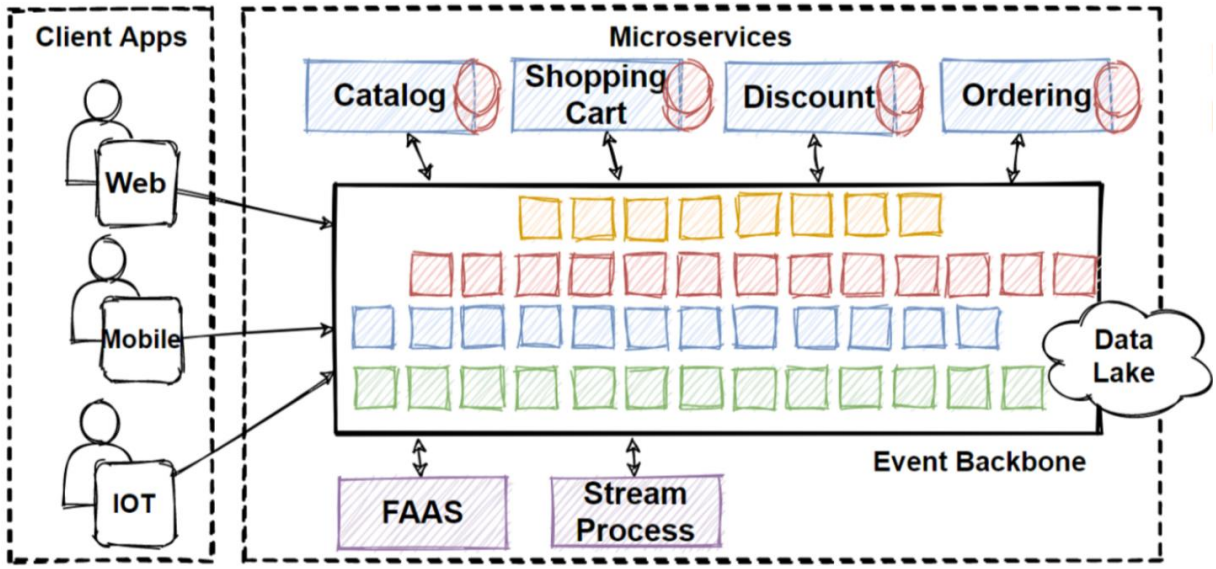
Event-Driven Approach with Asynchronous communication

- **Publish/subscribe** mechanism used in patterns like Event-driven microservices architecture
- **Event-bus** or **message broker system** is publishing events between multiple microservices
- Communication provide with **subscribing** these **events** in an async way.
- **Kafka** and **RabbitMQ** is the best tools for this operations. In Serverless world, **AWS EventBridge** is best option for async operations.
- **Fanout Pattern** Implementation with Amazon EventBridge and SQS



Event-Driven Microservices Architecture

- Communicating with microservices via **event messages**. Do **asynchronous behavior** and **loosely coupled** structures.
- **E-commerce** application **use cases** whichs are a customer create orders with some products and if the payment is successful, the products should be delivered to the customer.
- **Flow of events** like;
  - a customer creates an order
  - the customer receives a payment request
  - if the payment is successful the stock is updated and the order is delivered
  - if the payment in not successful, rollback the order and set order status is not completed.
- **Human readable** and if a new business requirement appears, it is easier to change the flow.



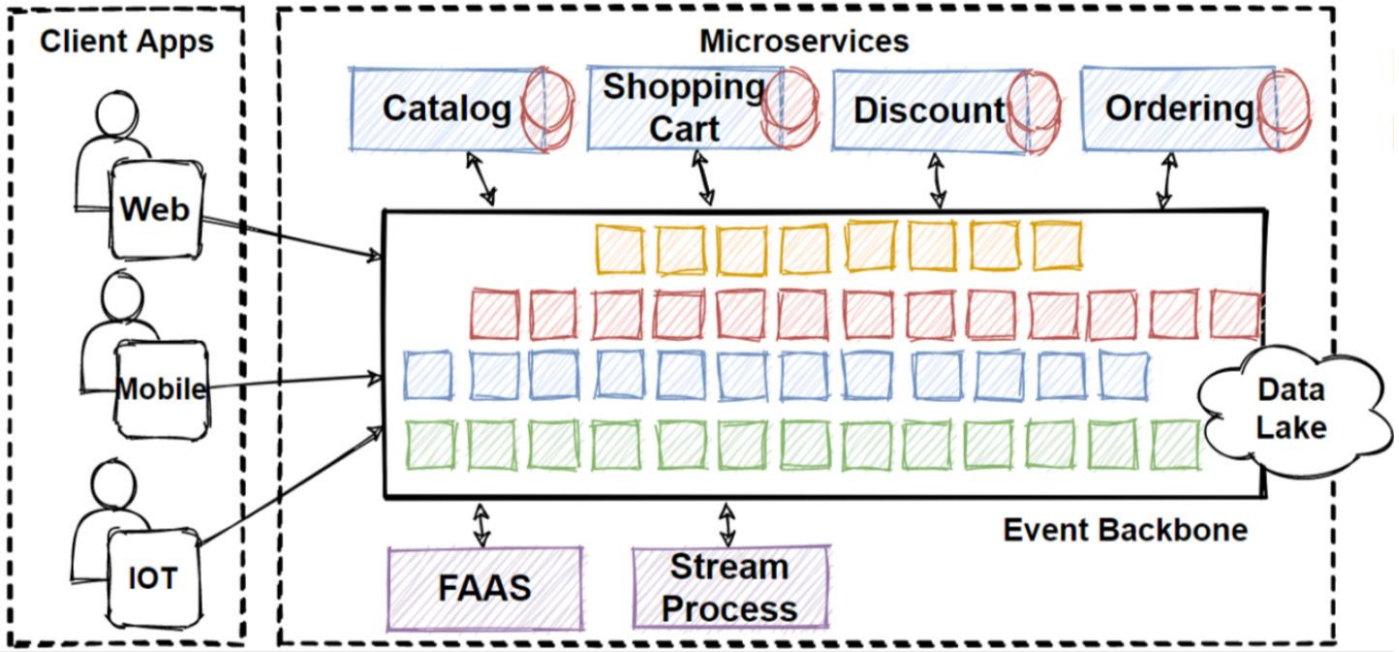
Event-Driven Microservices Architecture

- **Microservices** will only care about the **events**, not about the other microservices. they **process** only **events** and **publish** new **event** to **trigger** other services.
- Event-Driven Microservices Architectures like using **real-time messaging platforms**, **stream-processing**, **event hubs,** real-time processing, batch processing, data intelligence and so on.
- Communication via **Event-Hubs**. Think Event-Hubs is huge event store database that can make real-time processing.
- Every microservices, application, IOT devices,

even **FAAS serverless services** can interact

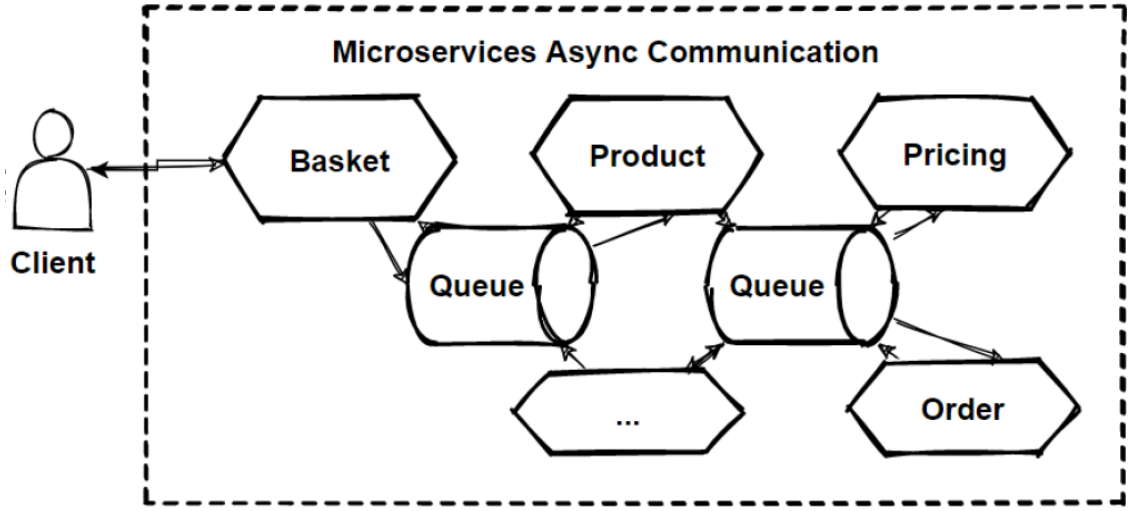
with each other with subscribing events

in **Event Hub.**

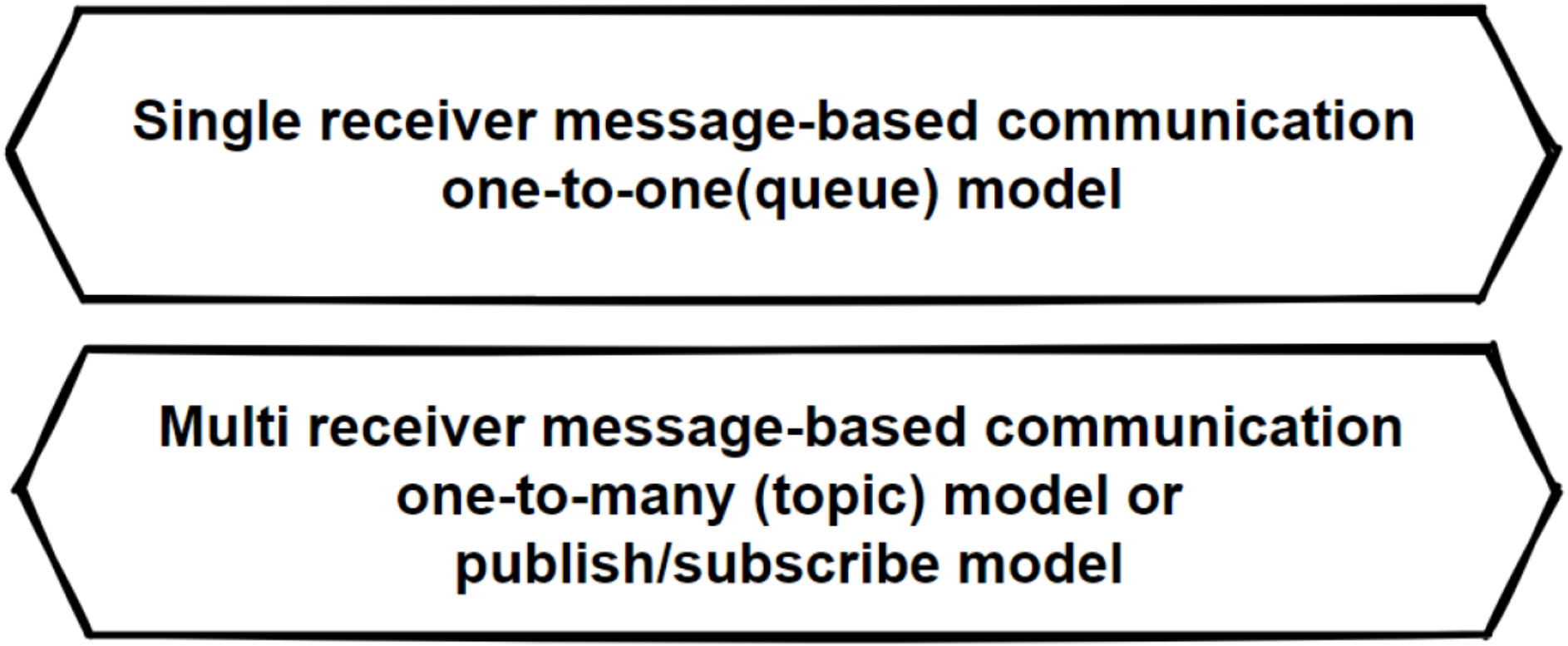


Application Integration Patterns for Microservices

- To **interact multiple microservices** without any dependency or make loosely coupled, follow Application Integration Patterns with using Asynchronous message-based communication
- **Events** can place the communication between microservices Event-driven communication.
- Any changes happens in the **Domains** of **microservices**, **propagating changes** across multiple microservices as an **event**, after that these events consumed by **subscriber** microservices.
- This event-driven communication and asynchronous messaging brings us "**eventual consistency**" principle.
- Send through **asynchronous** protocols like **AMQP** over the message broker systems like **Kafka** and **Rabbitmq**.

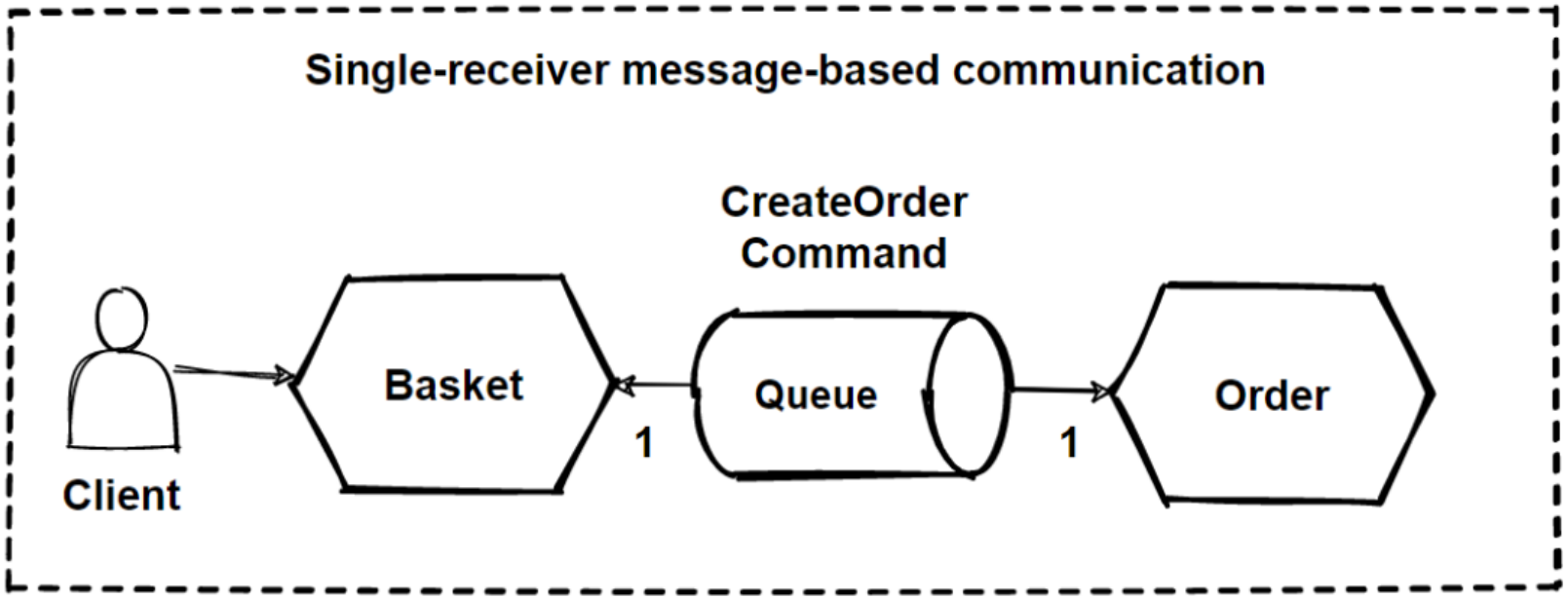


2 Type of Asynchronous Messaging Communication



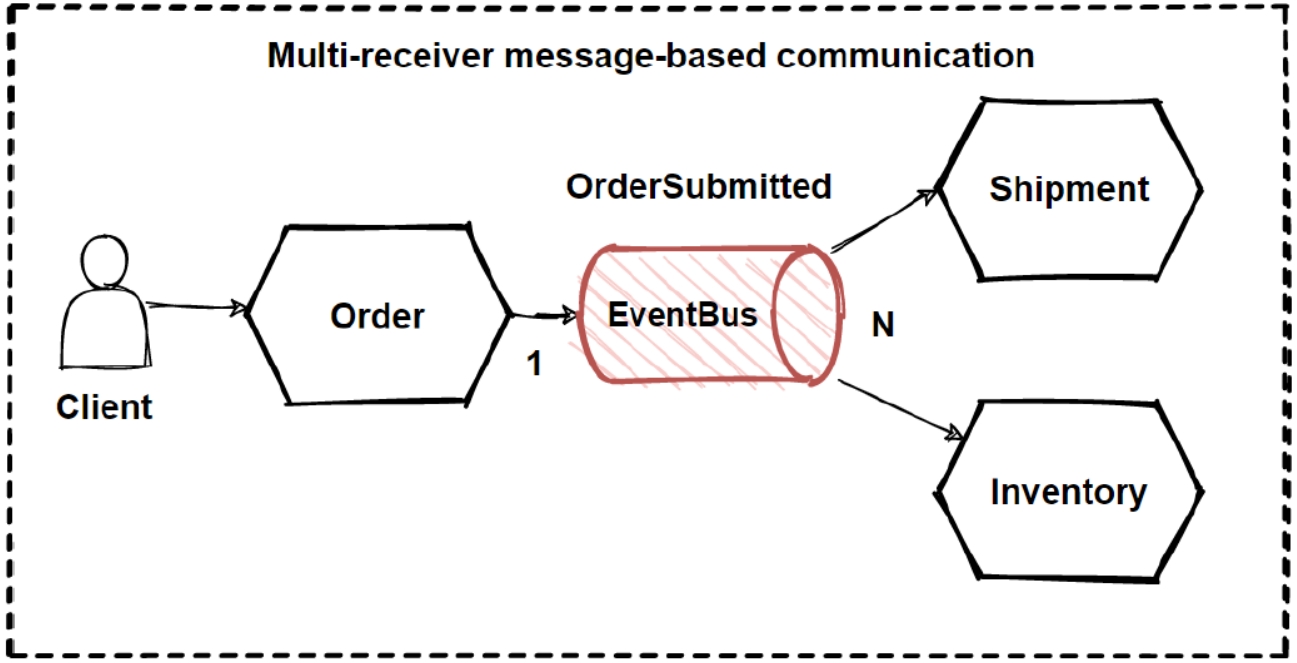
Single-receiver Message-based Communication

- **one-to-one** or **point-to-point** communications
- Without any dependency and make **loosely coupled**
- Example Use Case; **CreateOrder** request sends from the Basket microservices
- **CreateOrder** event



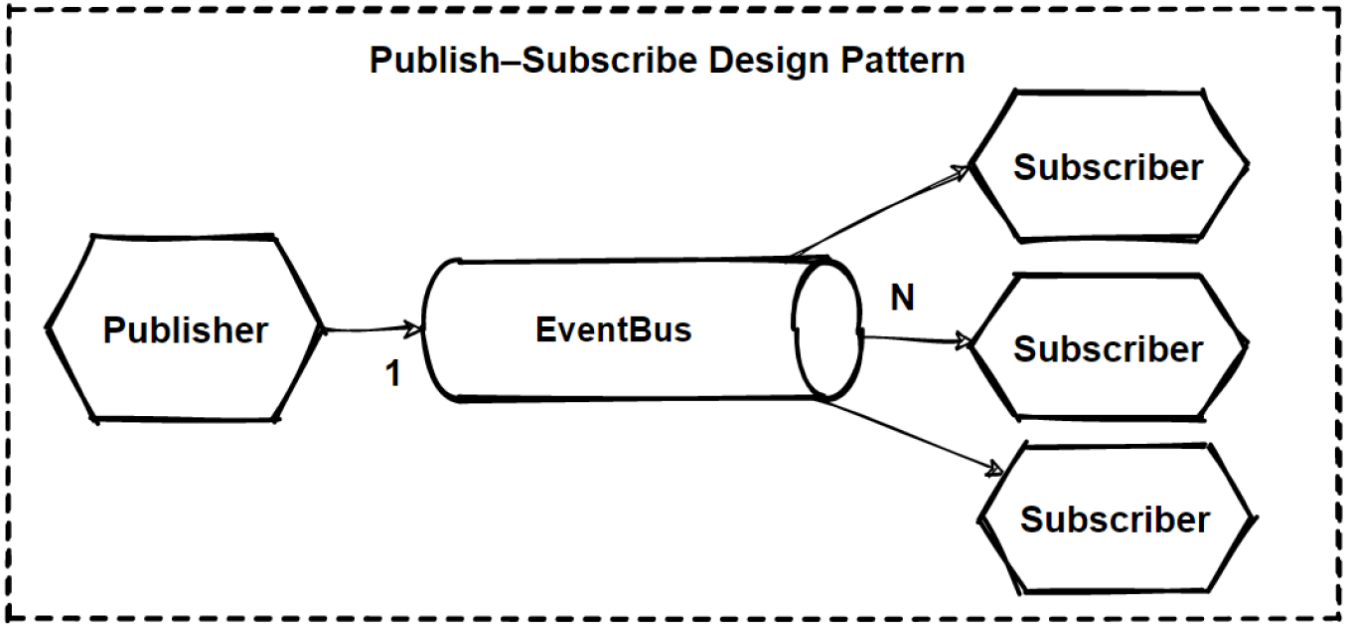
Multiple-receiver Message-based Communication

- **one-to-many** and **publish/subscribe** mechanisms**,** Consumer service **publish** a **message** and consumes from several microservices
- **Publish/subscribe** operations should require an **event bus** interface to publish events to subscribers.
- In **Asynchronous event-driven communication**, microservice publishes an event when something happens.
- **Order microservices** publishes **OrderSubmitted** events to an event bus and **Shipment** and **Inventory** microservices can subscribe to this event.



Publish–Subscribe Design Pattern

- Sender of messages are called **publishers**, and has specific receivers are called **subscribers**.
- publishers don’t send the messages directly to the subscribers.
- Publishers and subscribers communicate each other, without **coupling** or any **dependency** of each other.
- **Decouples microservices** communications, so that microservices can be managed and **scale independently**.
- Increases scalability and improves **responsiveness** of the system.



Thinking AWS Serverless for Event-Driven Microservices Architecture

→ Moving Event-Driven Microservices Architecture with thinking AWS Serverless Services.

What is Serverless ?

- **Build** and **run** applications **without thinking about servers** - running code, managing data, and integrating applications, all **without managing servers**.
- **Automatic scaling**, built-in **high availability**, and a **pay-for-use billing model** to increase agility and optimize costs.
- AWS Lambda, an **Event-driven compute** service natively integrated with over 200 AWS services
- Focus on the building application.
- **4 main pillars**
  - No Infrasturcture provisioning, no management
  - Pay for value
  - Automatic scaling
  - High available and secure

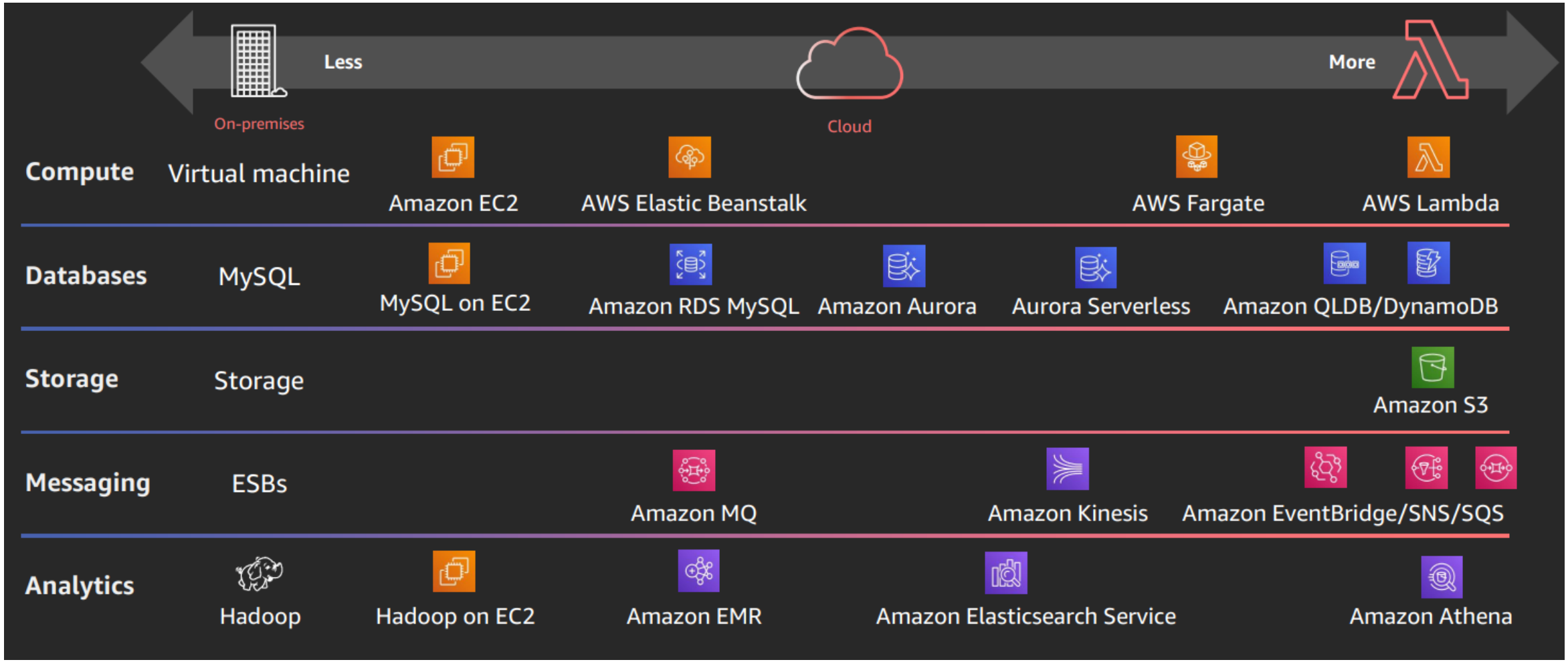
Benefits of Serverless

- **Move from idea to market, faster**
  Eliminate operational overhead so your teams can release quickly, get feedback, and iterate to get to market faster.
- **Lower your costs**
  With a pay-for-value billing model, resource utilization is automatically optimized and you never pay for over-provisioning.
- **Adapt at scale**
  With technologies that automatically scale from zero to peak demands, you can adapt to customer needs faster than ever.
- **Build better applications, easier**
  Serverless applications have built-in service integrations, so you can focus on building your application instead of configuring it.

Thinking Serverless

- **Features First**
  Breaking down the application to simple components that allows to think small with serverless. We can follow "decompose by business capacity" principle.
- **Focus on Events**
  Events is the key of serverless applications. A serverless solution is entirely event-driven.
- **Stateless**
  This is key to on-demand scaling, so with event-driven architectures, events are immutable objects publish and subscribe from stateless services.
- **Data Flow**
  AWS provides really good amount of different data services that you can handle your data problems in serverless design.
- **Use the services**
  AWS has broad range of different tools that you can use when designing your systems.

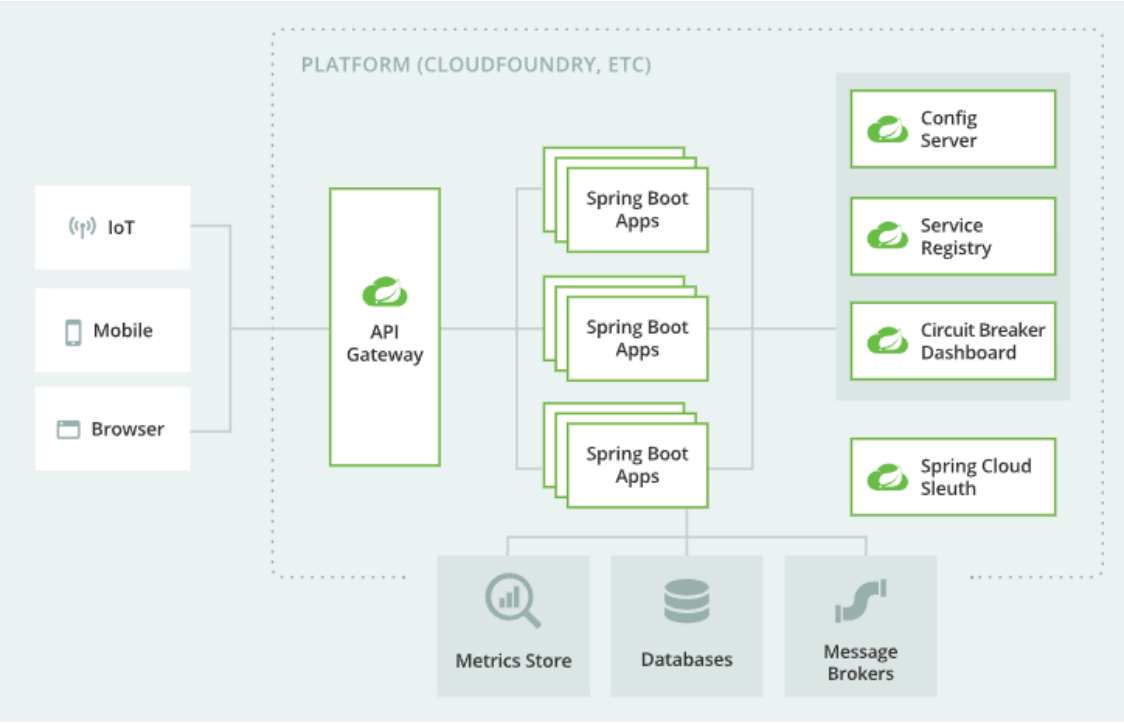
Thinking Serverless



https://d1.awsstatic.com/events/reinvent/2019/REPEAT_3_Serverless_architectural_patterns_and_best_practices_ARC307-R3.pdf

What is Application Development Framework

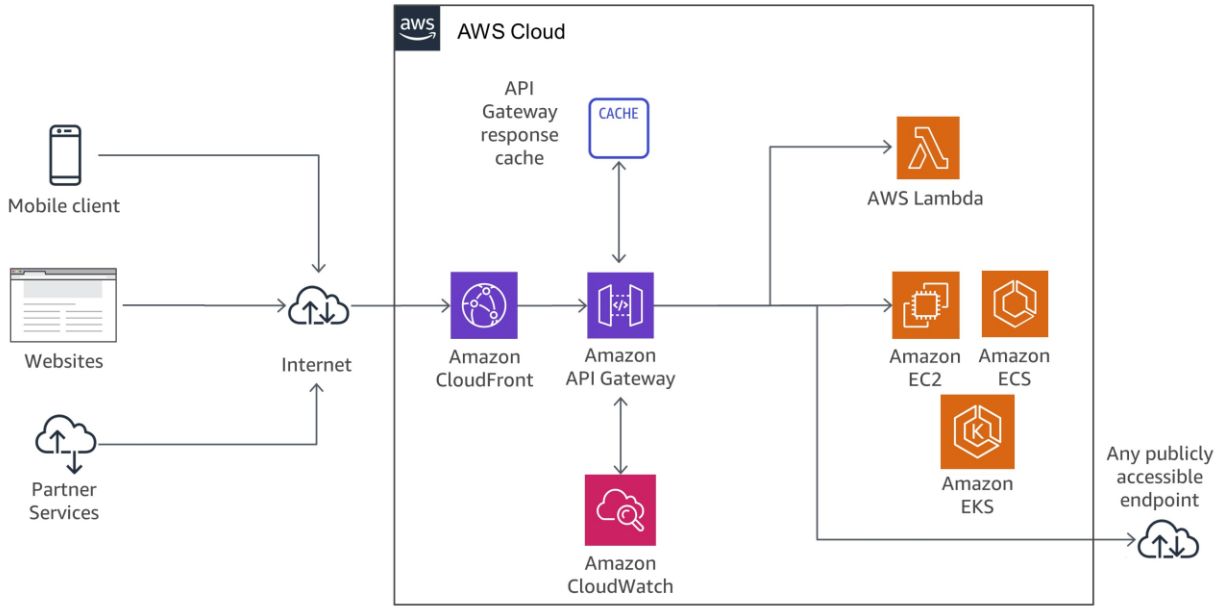
- **Application Development Framework** is a set of code that is responsible for calling your actual business logic based on its defined architecture.
  - to create **RESTful APIs** with underlying web application
  - to **connect databases** with ORM tools
  - to provide centralized **logging** features
  - to provide **Identity** Management
  - to collect and visualize **monitoring** and **metrics**
  - to facilitate to connect 3rd party services like Kafka, Redis, RabbitMQ so on..
- Provides all those features in order to focus on actual business logic.
- **Java Spring Boot** or **.Net ecosystem** as an Application Development Framework.
- Have to develop all those stuff by yourself.



https://spring.io/microservices

AWS as an Application Development Framework

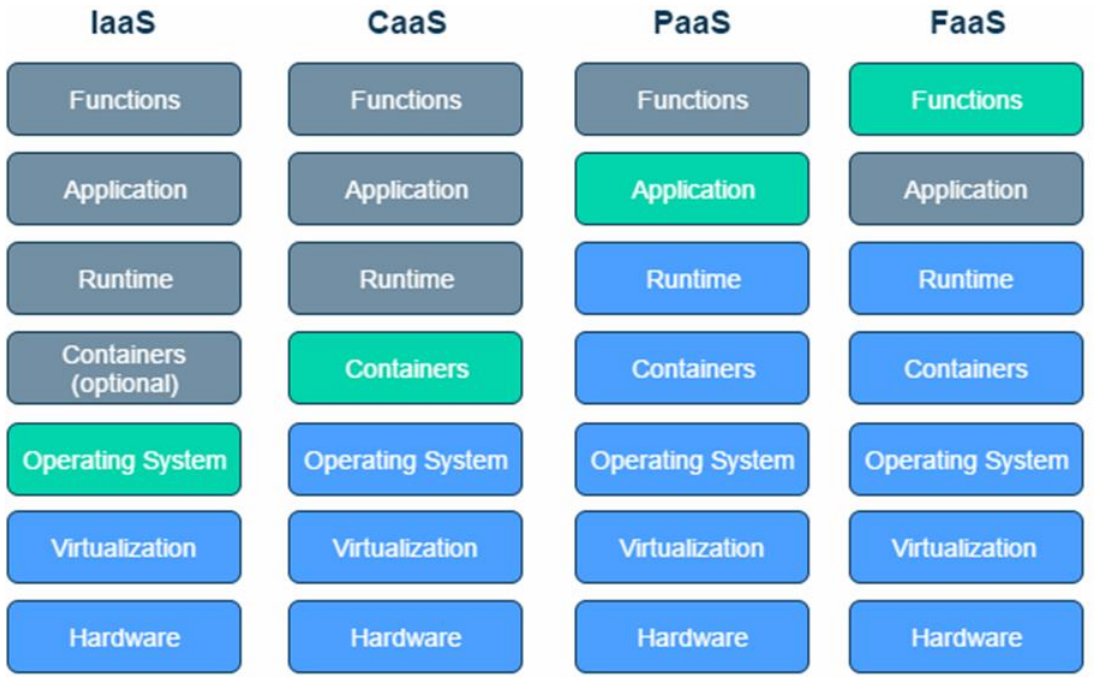
- All these features already provided by **AWS Serverless Services** without thinking **scalability**, **availability** and no configuration code requirement.
- Focus on building actual **business logic** and no worry about periphery requirements
- Serverless applications have **built-in service** integrations, more than **200 services** easy to integrate each other to provide value of business
- Automatically **scale** from **zero** to **peak** demands, you can adapt to customer needs faster than ever.
- **AWS** has **overwhelming advantage** when we compare with other **Application Development Frameworks**.



https://docs.aws.amazon.com/whitepapers/latest/microservices-on-aws/api-implementation.html

Evolution of Cloud Infrastructures

- **Infrastructure as a Service (IaaS)**
  Renting a computer through a cloud provider. all control, including the operating system level, is yours.
- **Container as a Service (CaaS)**
  Cloud provider puts another level of abstraction here so that you can easily manage your containers.
- **Platform as a Service (PaaS)**
  Upload your program after choosing the language/framework in which your program will run.
- **Functions as a Service (Faas) — Serverless**
  Run as a result of an incoming event and our application closes after the function finishes its work. Scaling of the application, deployment, operating system or programming language updates not entirely our problem.



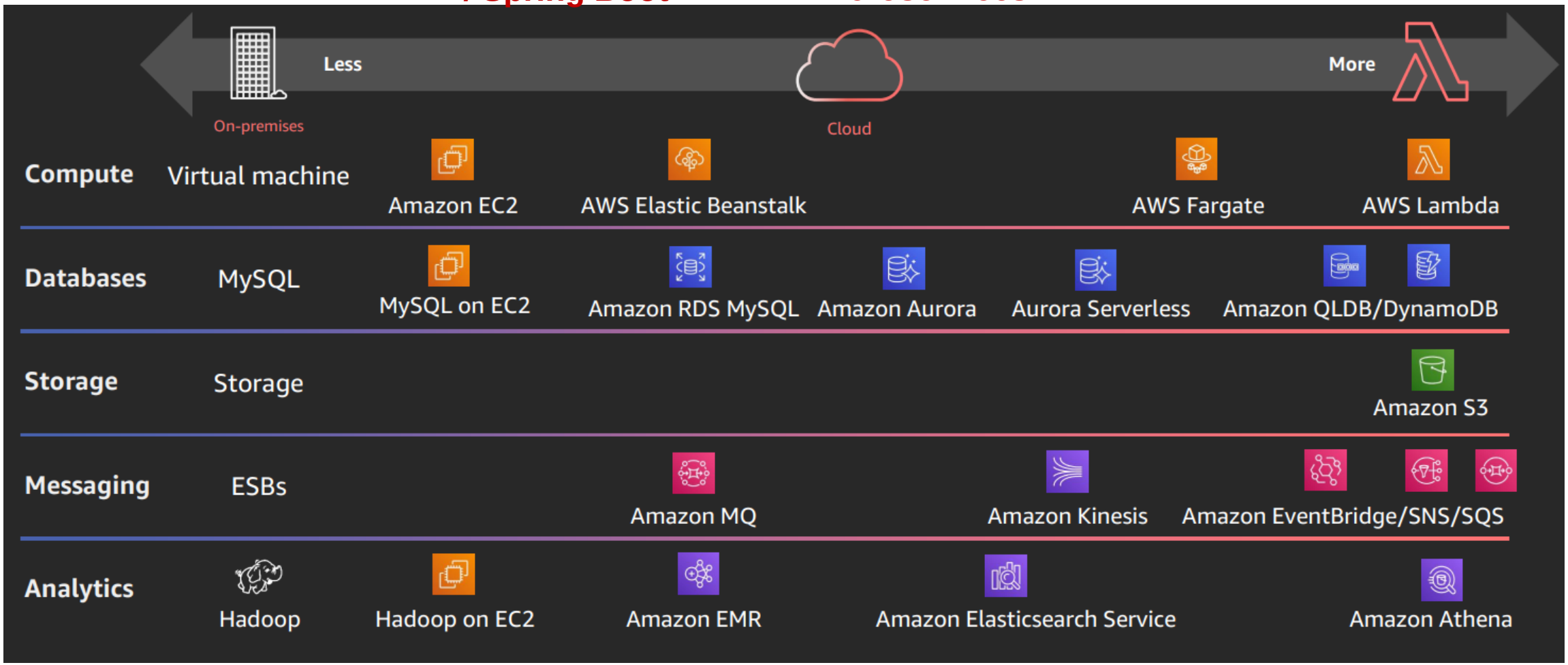
https://serverless.zone/abstracting-the-back-end-with-faas-e5e80e837362

2000s – Java / .Net

2010s – Rails / Django / Spring Boot

2015s – Containers, Microservices

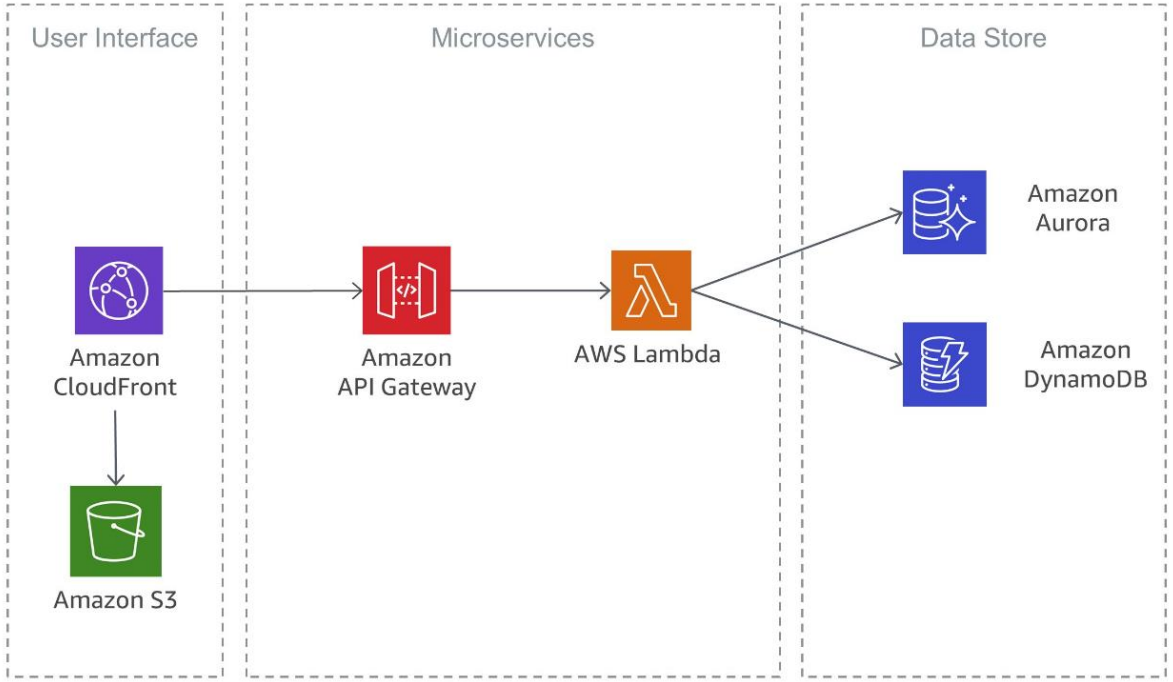
2020s – AWS Serverless Framework



https://d1.awsstatic.com/events/reinvent/2019/REPEAT_3_Serverless_architectural_patterns_and_best_practices_ARC307-R3.pdf

AWS Lambda as a Microservice

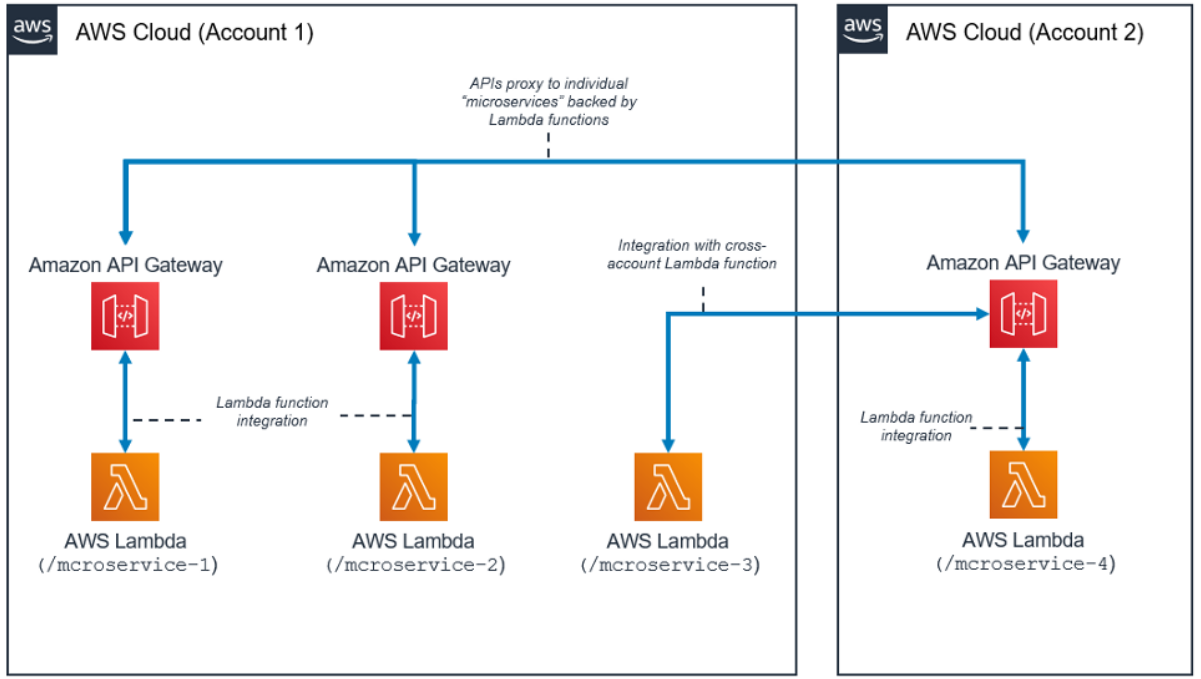
- Microservice are **small business services** that can work together and can be deployed **autonomously** / **independently**.
- **Lambda** is a service that allows you to run your functions in the cloud **completely serverless** and eliminates the operational **complexity**.
- It **integrates** with the **API gateway**, allows you to invoke functions with the API calls, and makes your architecture completely serverless.
- Microservice with **AWS Lambda**, which removing the architectural overhead of designing for **scaling** and high **availability**,
- Eliminating the **operational efforts** of operating and monitoring the microservice's underlying infrastructure.



https://docs.aws.amazon.com/whitepapers/latest/microservices-on-aws/serverless-microservices.html

Serverless Microservices with Lambda

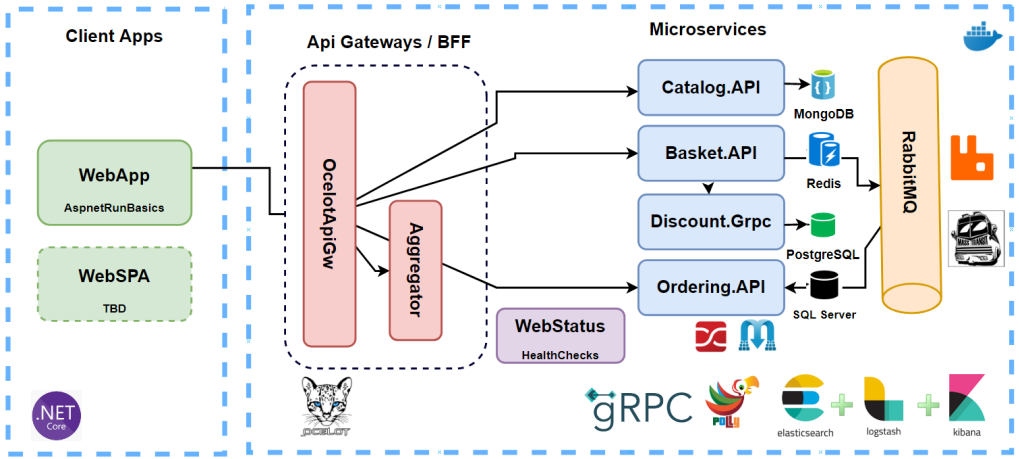
- Each of the application components is **decoupled** and **deployed** and run **independently**.
- AWS **Lambda-initiated functions** is all you need to build a microservice.
- A microservices environment can introduce
  - **repeated overhead** for create each new microservice,
  - **problems optimizing** server usage,
  - **complexity** of running multiple versions of microservices,
  - client-side code **requirements** to **integrate** with many services.
- **Serverless microservices pattern** reduces the barrier for the creation of each subsequent microservice
- Optimizing **server utilization** is no longer relevant with this pattern.



https://docs.aws.amazon.com/whitepapers/latest/microservices-on-aws/serverless-microservices.html

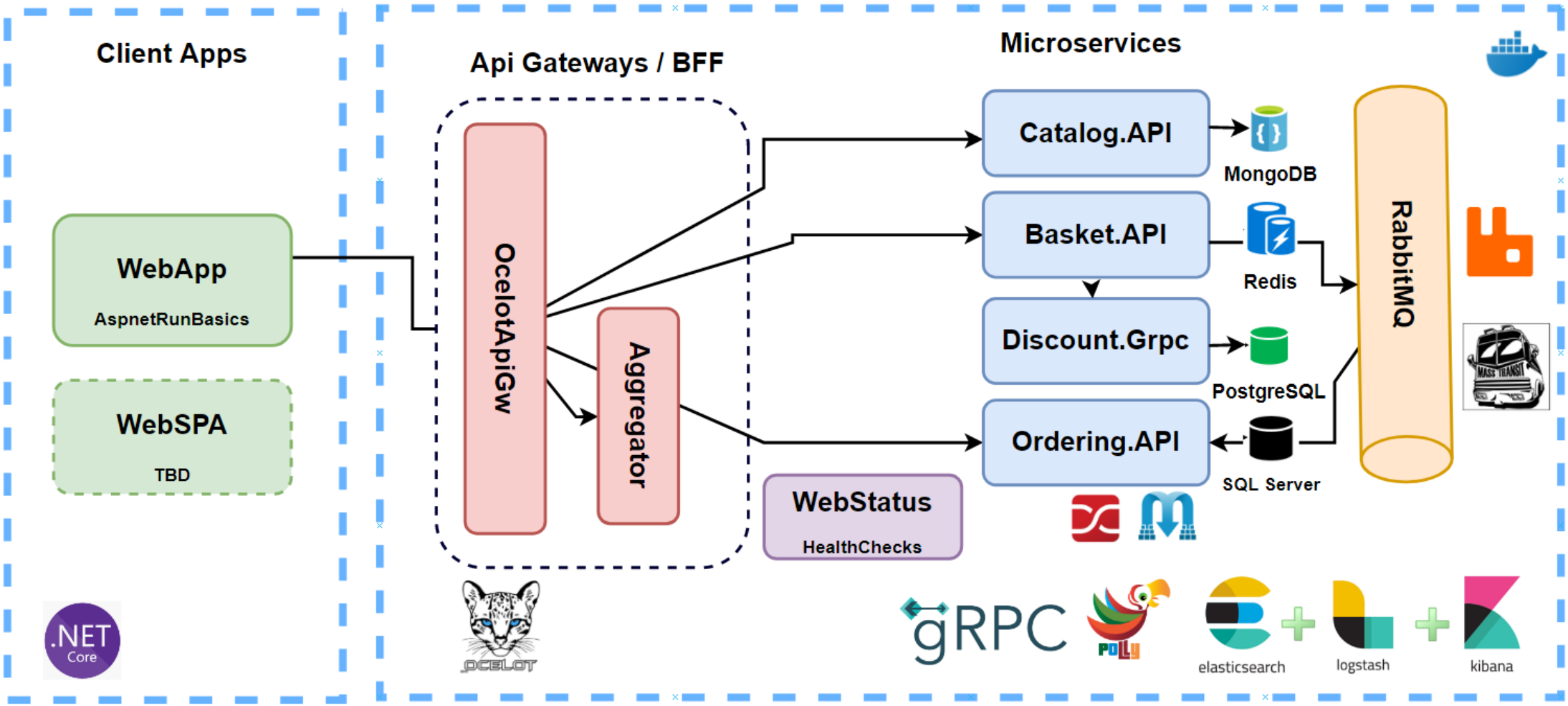
Cloud-Native E-Commerce Microservice Architecture

- Developed in cloud-native environment
  - developing with application development frameworks like java spring boot or .net
  - containerize by docker containers
  - container orchestrate with k8s and deployed to k8s
  - and IaC by Terraform or any cloud-native infrastructure developing tool.
- **Custom api gateways** developed by libraries like ocelot, kong.
- **Microservices** developed with development frameworks like java spring boot or .net
- **Databases** both relational and no-sql key-value pair or document databases using mongodb, redis, postgres and sql server.
- **Message broker system** which is Rabbitmq or Kafka in order to provide async communication between microservices.
- **Elastic search + Logstash + Kibana** for centralized logging and monitoring



https://github.com/aspnetrun/run-aspnetcore-microservices

Cloud-Native E-Commerce Microservice Architecture

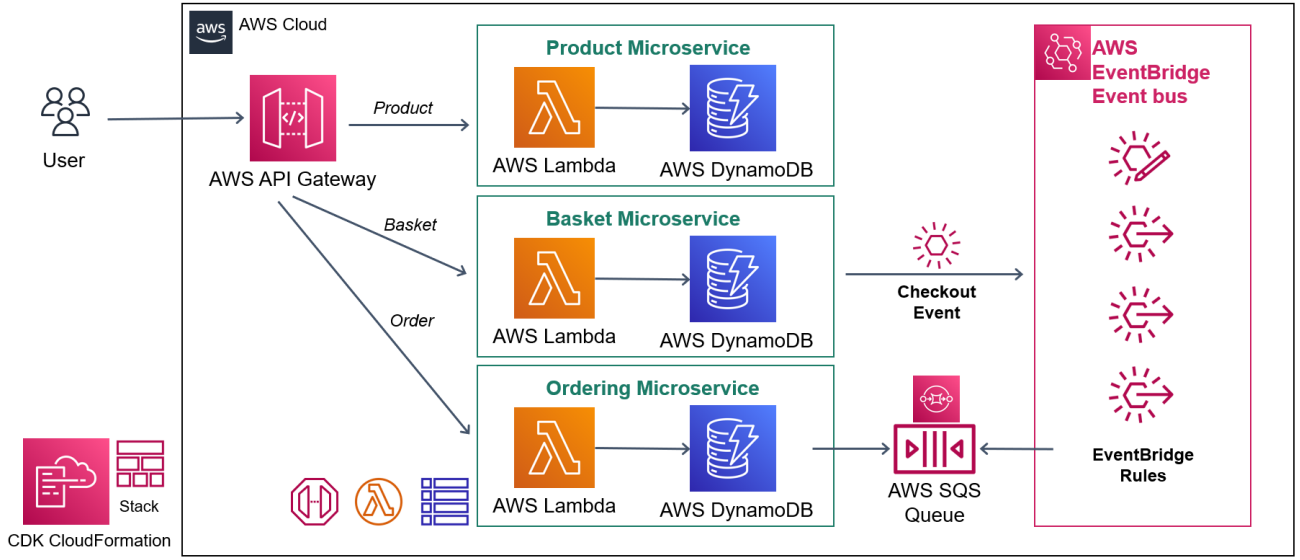


https://github.com/aspnetrun/run-aspnetcore-microservices

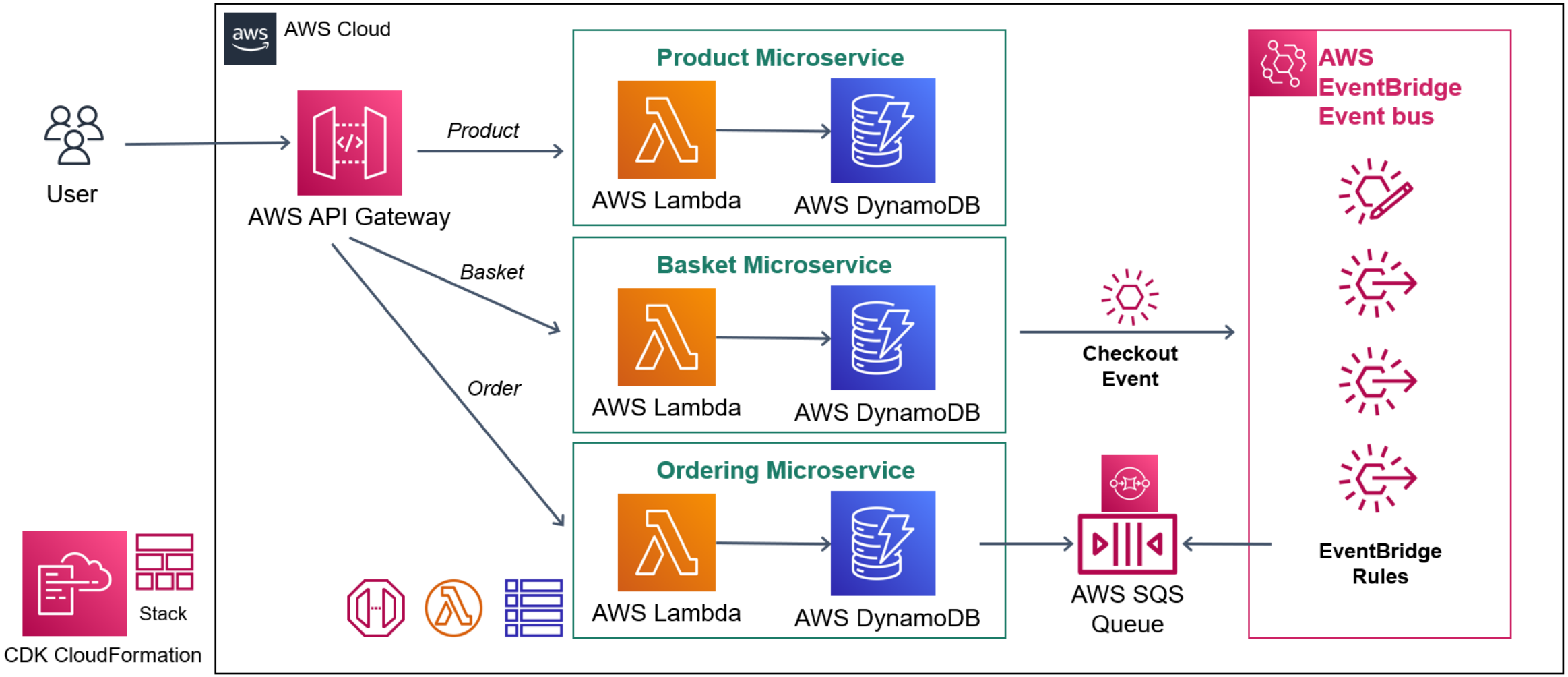
Serverless E-Commerce Microservice Architecture

- Developed in Serverless environment
  - developing with AWS Lambda
  - No Deployment Requirement
  - and IaC by AWS CDK CloudFormation
- **AWS API Gateway**
- **Microservices** developed with AWS Lambda
- **Databases** are no-sql DynamoDB that can be

  key-value pair or document databases
- **Message broker system** which is Amazon EventBridge Eventbus
- **Amazon CloudWatch** for centralized logging and monitoring
- By default automatic scalability and high avaliability



Serverless E-Commerce Microservice Architecture

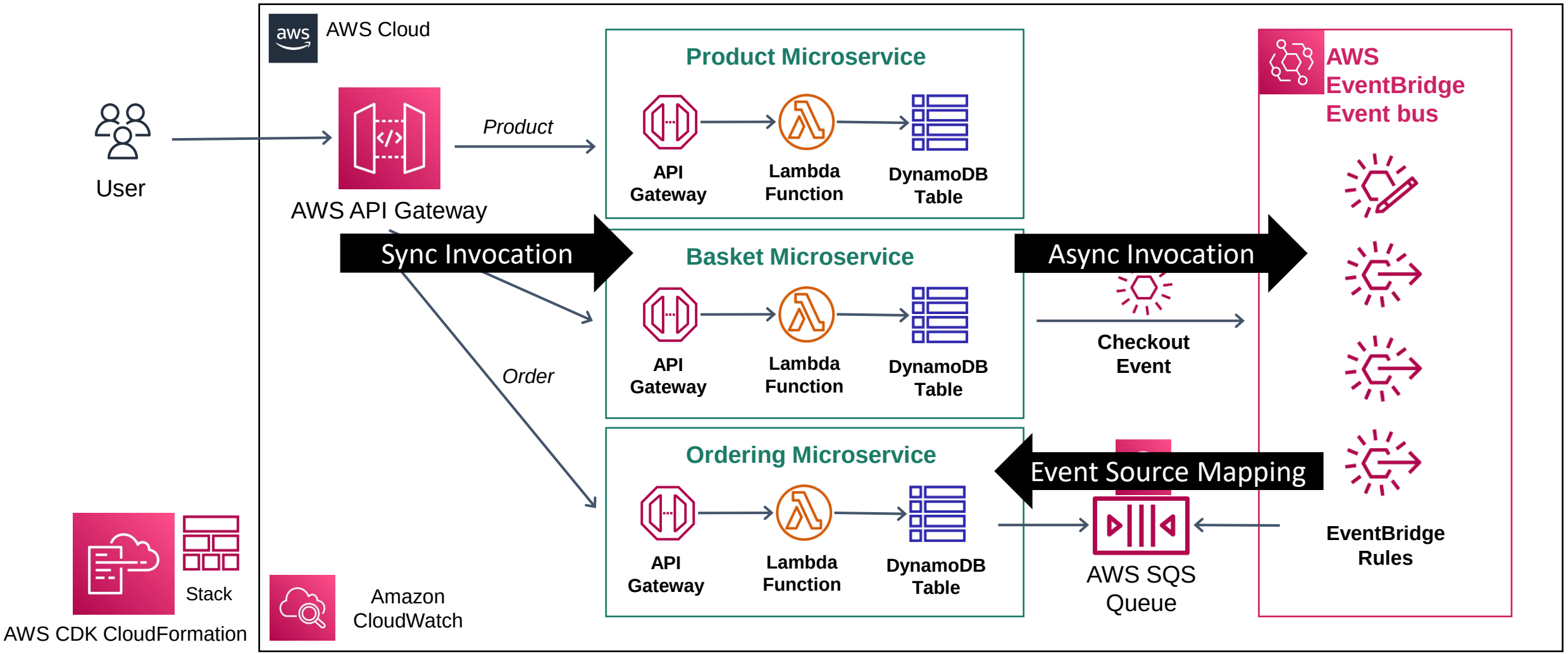


Event-driven Serverless Microservices Design with AWS Lambda

- **Event-driven architecture** provides developers with increased availability, elasticity, scalability, and cost optimization.
- **Lambda** is very good fit with **event-driven architectures**, because the nature of lambda executions trigger from events.
- Events can come **lots of resources** and able to trigger lambda functions.
- An **event triggering a Lambda function** could be almost anything.
- The event **publisher system is unaware** of which consumers are subscribing that event.
- **Lambda-based** applications use a **combination of AWS services**, providing business logic to transform data that moves between services.

https://docs.aws.amazon.com/lambda/latest/dg/invocation-eventsourcemapping.html

AWS Serverless Microservices with AWS Lambda Invocation Types



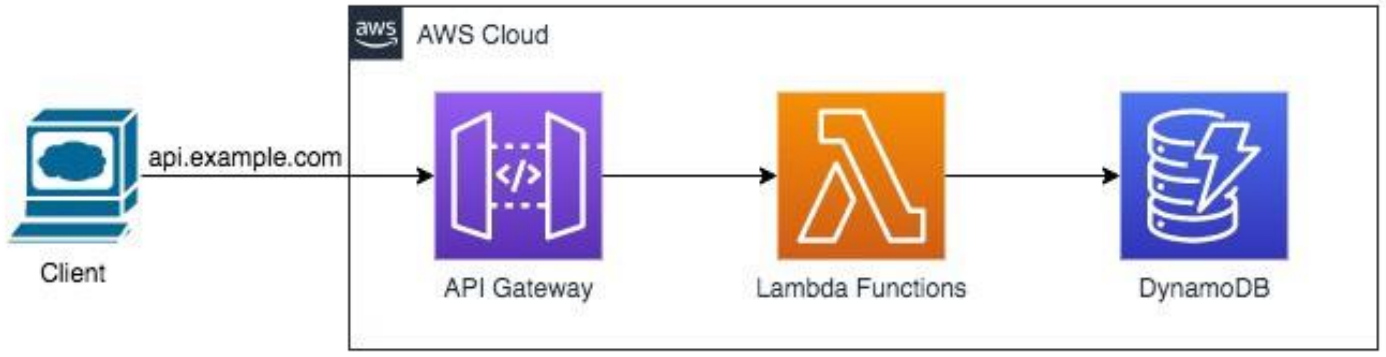
Serverless + CDK Automation + Integration Patterns = AWSome!

- **AWS Serverless** will be our application development framework.
- **AWS CDK** is IaC tool that we will develop whole infrastructure with typescript coding
- **Integration Patterns** with Queue-Chaning, Publish-Subscribe and Fan-out design patterns
- AWSome Microservices !

https://www.youtube.com/watch?v=ttJAIQf7cTw

RESTful Microservices Pattern

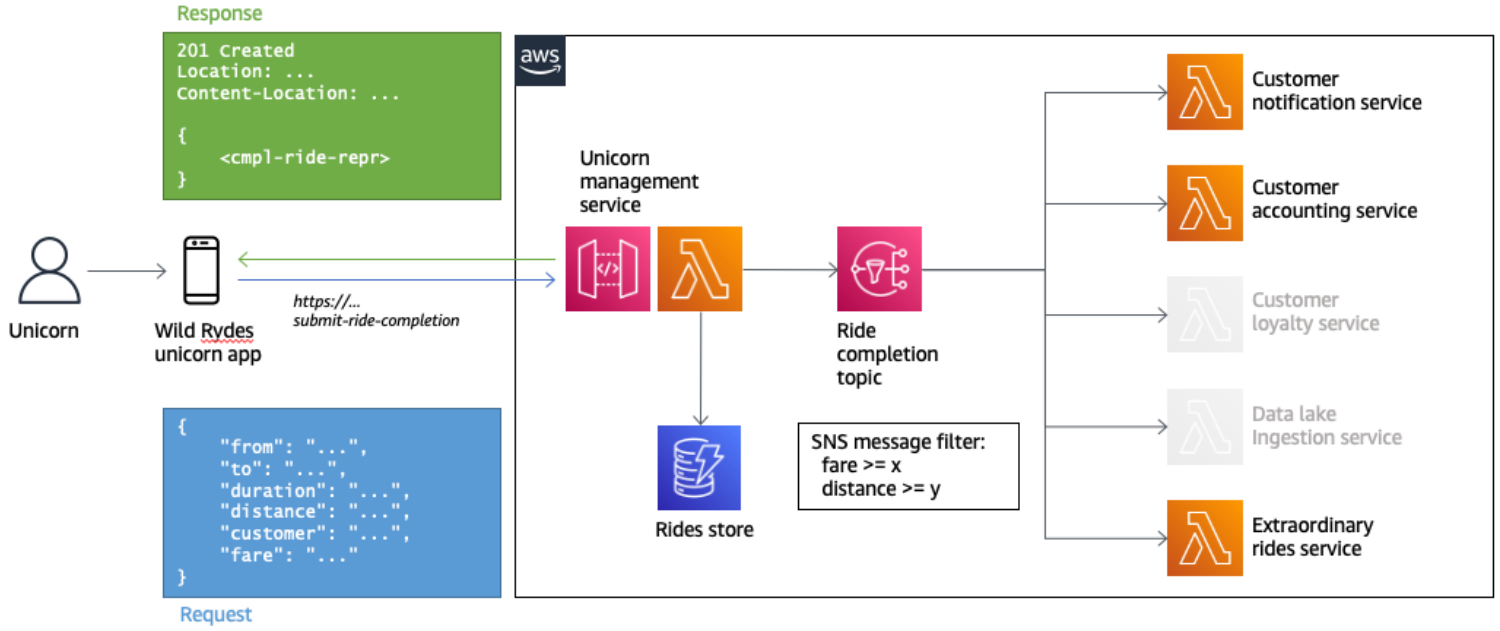
- **Sync Communication** for RESTful Microservices
- Synchronous commands are **request / response**.
- Restful Microservices with **API Gateway, Lambda Functions** and **Dynamodb**.
- REST API and CRUD endpoints (AWS Lambda, API Gateway)
- Data persistence (AWS DynamoDB)
- Restful API-Driven Development for **performing CRUD** operations among Lambda and Dynamodb.



Fan-Out & Message Filtering with Publish/Subscribe Pattern

- **Async Communication** for performing one-to-many and publish/subscribe mechanisms.
- Client service **publish a message** and it **consumes** from **several microservices** which's are subscribing this message on the message broker system.
- **Decouples Messaging** between services, building loosely-coupled architectures.
- Using in **event-driven** architectures
- **Publishes** an event something happen
- Price change in a product microservice
- **Subscribed** from SC microservice

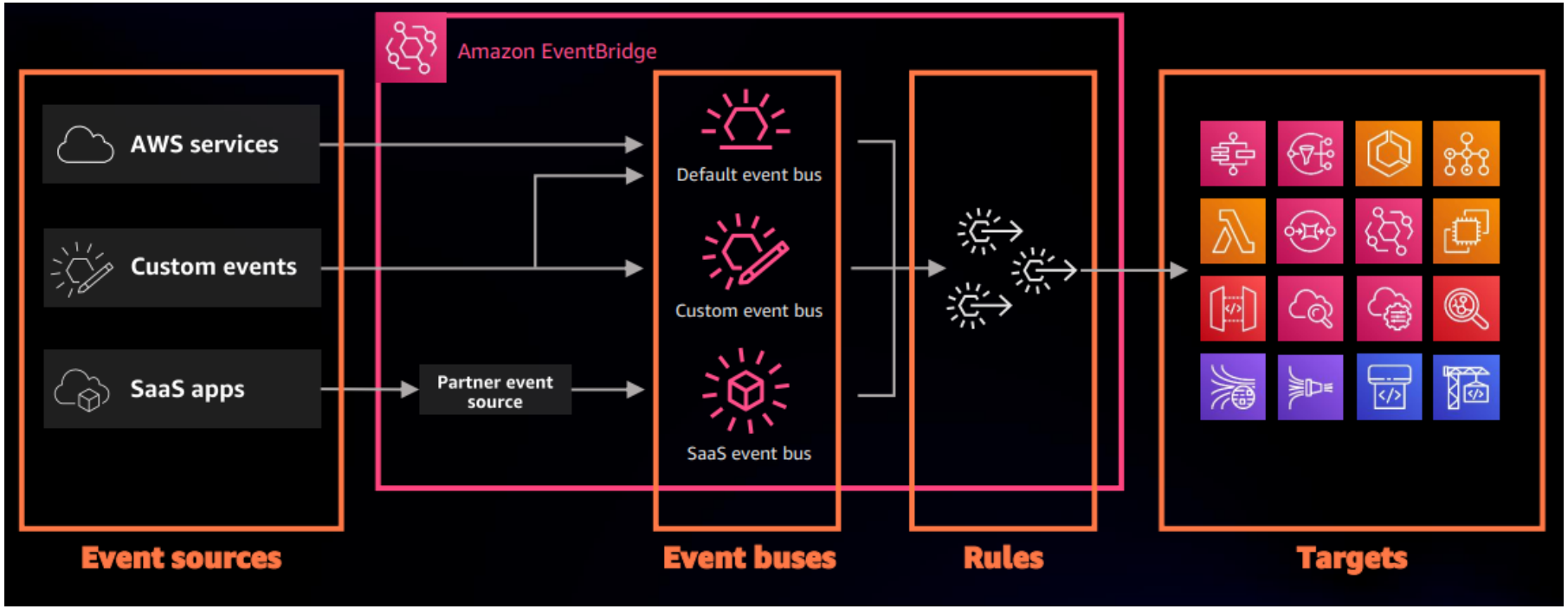
  to update basket price



https://async-messaging.workshop.aws/fan-out-and-message-filtering.html

Publish/Subscribe Pattern with Amazon EventBridge

- **Amazon EventBridge** for Event-Driven asynchronous Communication between Microservices
- Event Sources
- Event Buses
- Rules
- Targets



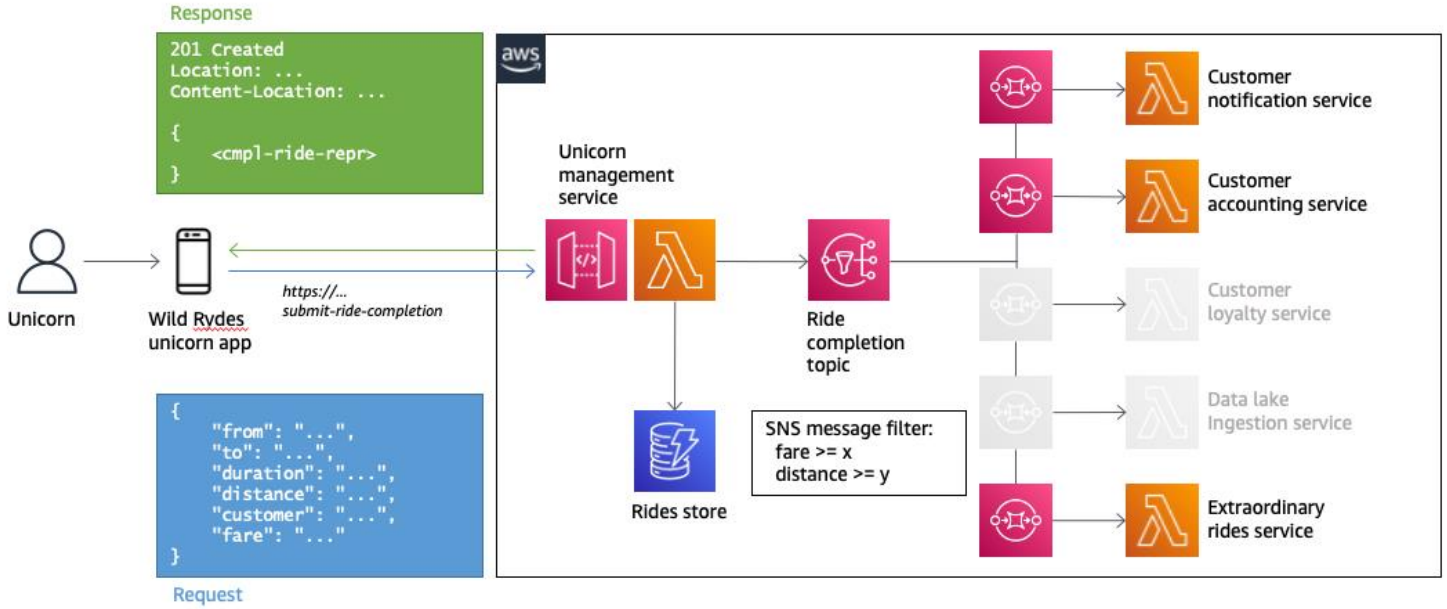
https://da-public-assets.s3.amazonaws.com/serverlessland/pdf/2021+-+Serverlesspresso+exhibit+-+PDF.pdf

Topic-Queue Chaining & Load Balancing Pattern

- Use a **queue** that acts as a **buffer** between the service to a**void loss data** if the service to fail.
- **Services** can be **down** or **getting exception** or taken offline for maintenance, then **events** will be **loses**, **disappeared** and can't process after the subscriber service is up and running.
- Put **Amazon SQS** between **EventBridge** and **Ordering** microservices.
- Store this event messages into **SQS queue**

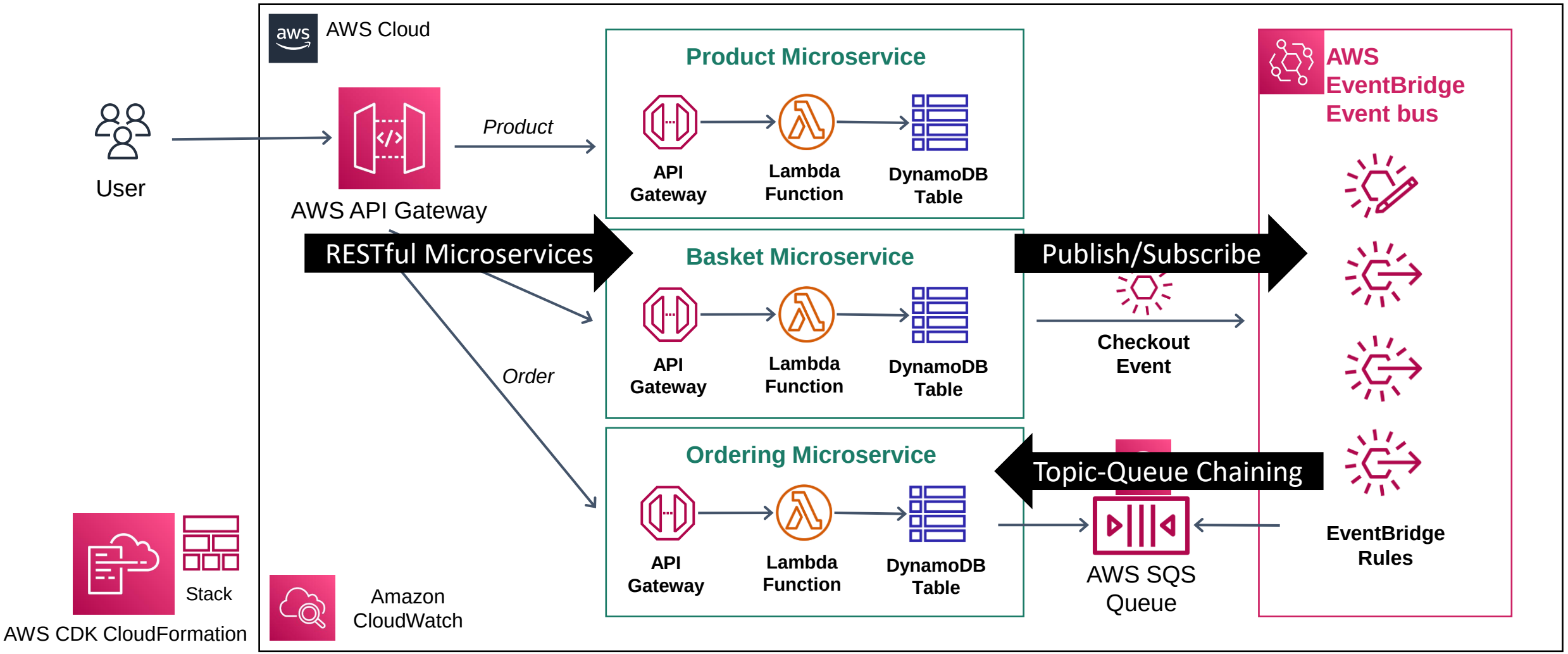
  with **durable** and **persistent** manner,

  no message will get lost
- **Queue** can act as a buffering **load balancer**



https://async-messaging.workshop.aws/fan-out-and-message-filtering.html

Serverless Patterns for Microservices



Serverless + CDK Automation + Integration Patterns = AWSome!

- **AWS Serverless** will be our application development framework.
- **AWS CDK** is IaC tool that we will develop whole infrastructure with typescript coding
- **Integration Patterns** with Queue-Chaning, Publish-Subscribe and Fan-out design patterns
- AWSome Microservices !

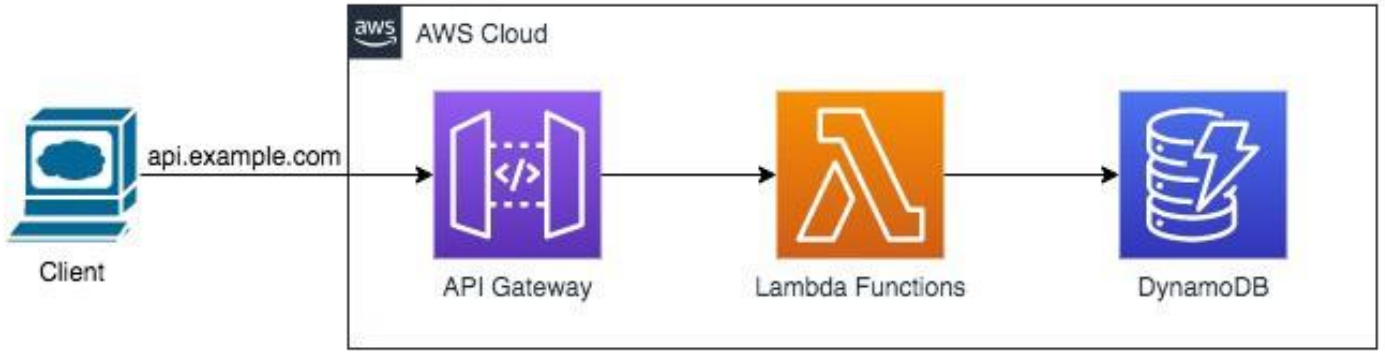
https://www.youtube.com/watch?v=ttJAIQf7cTw

API Gateway Restful API Development with Synchronous Invocations

→ API Gateway Restful API Development with Synchronous Lambda Event Sources.

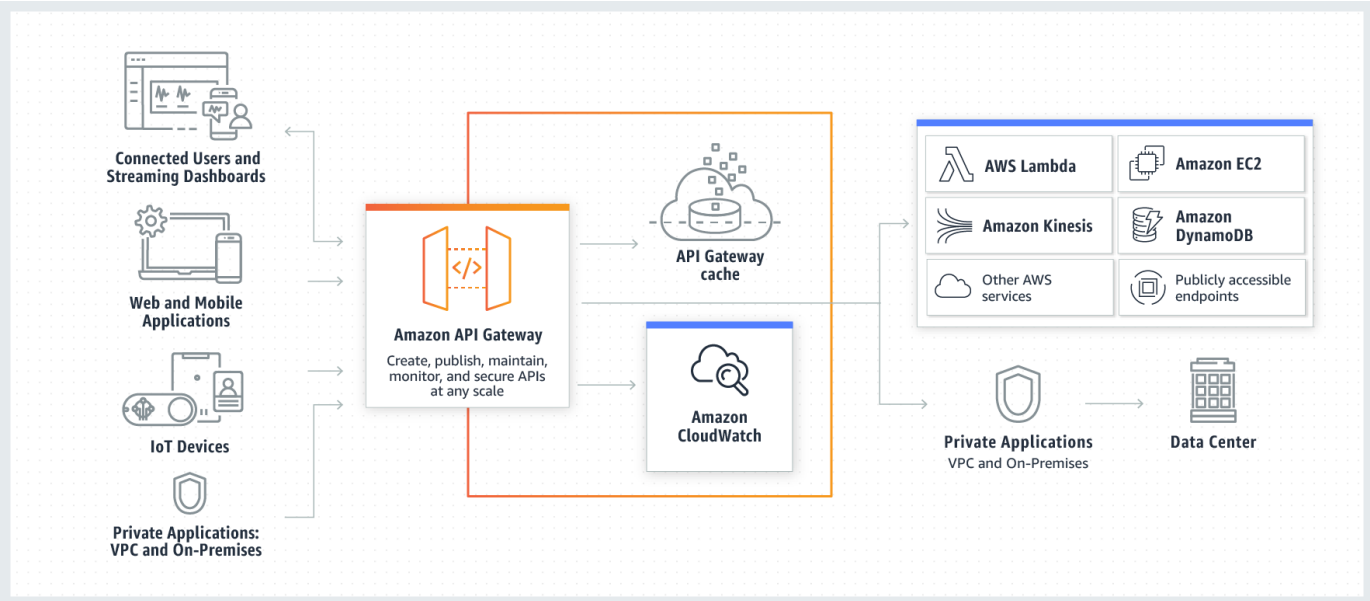
API Gateway Restful API Development with Synchronous Lambda Event Sources

- What is Amazon API Gateway?
- Architecture of API Gateway
- Main Features of API Gateway
- Amazon API Gateway Use Cases
- API Gateway as a Lambda Synchronous Event Sources
- Amazon API Gateway Core Concepts
- Amazon API Gateway -Differences between REST - HTTP API
- Amazon API Gateway Walkthrough with AWS Management Console



What is Amazon API Gateway?

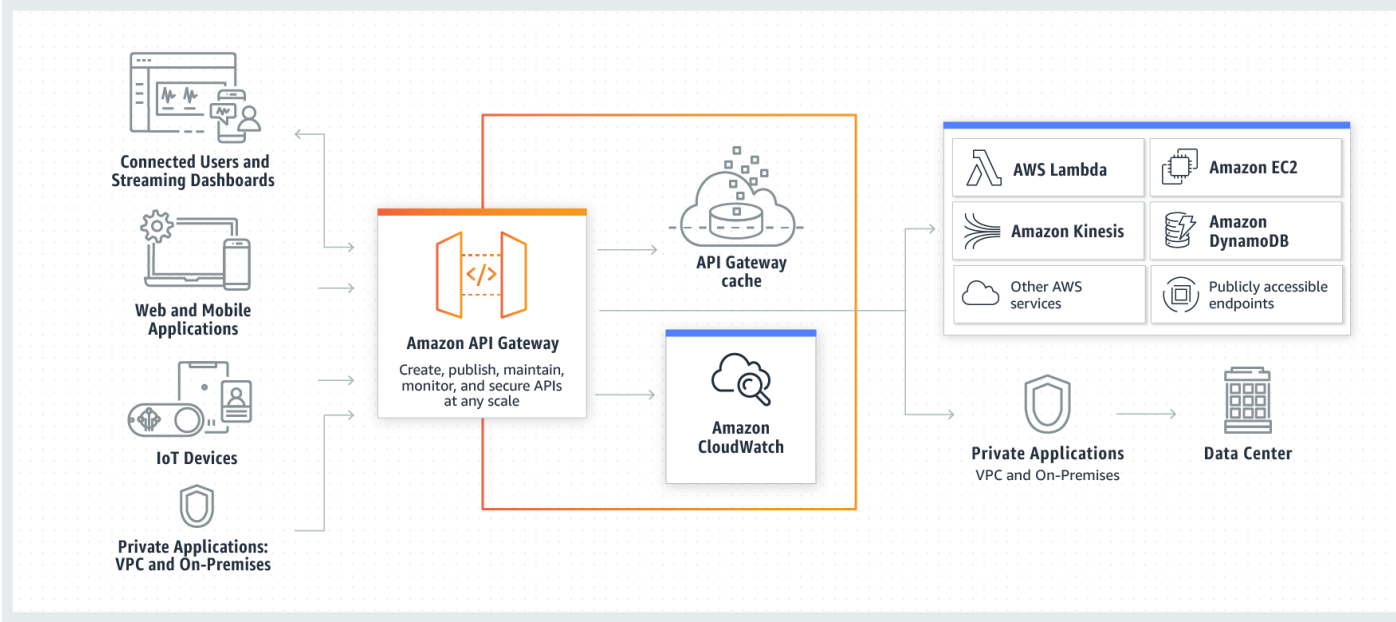
- **Fully managed** service for developers to create, publish, maintain, monitor and secure APIs at any scale.
- **Front door** for applications to access data, business logic from your backend services.
- Create **RESTful APIs** and **WebSocket APIs**
- **RESTful APIs** expose backend HTTP endpoints, AWS Lambda functions, or other AWS services.
- **RESTful APIs** optimized for serverless workloads and HTTP backends using HTTP APIs.
- **WebSocket APIs** are real-time two-way communication
- **Expose microservices** with RESTful APIs



https://aws.amazon.com/api-gateway/

Architecture of API Gateway

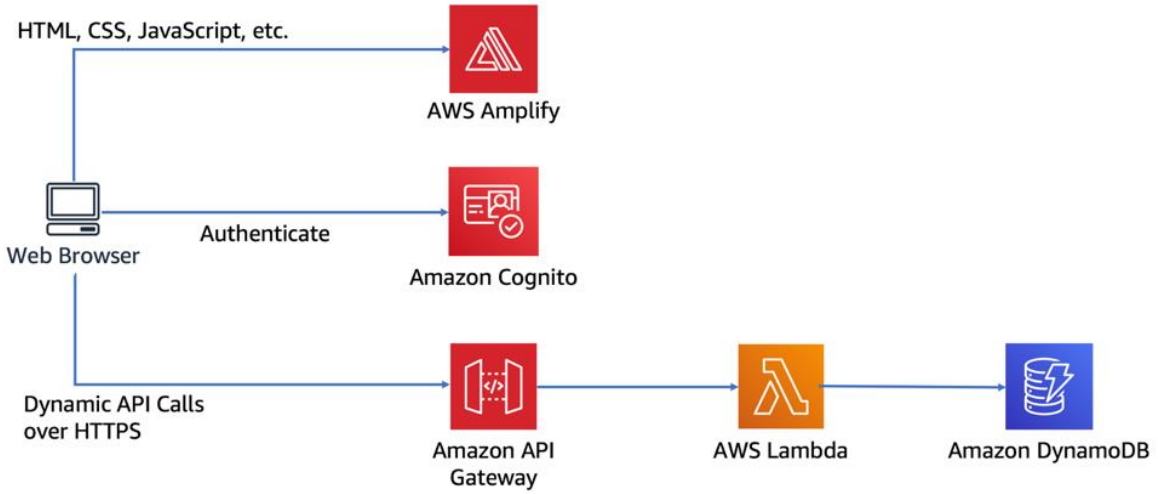
- Provide your customers with an **integrated** and **consistent** developer experience.
- Handles tasks in **accepting** and **processing** hundreds of thousands of **concurrent** API calls.
- Tasks; **traffic** management, **authorization** and access control, **monitoring**, and **API version** management.
- **Front door** for applications to access data, business logic, or functionality from your backend services.
- **Expose microservices** with RESTful APIs



https://aws.amazon.com/api-gateway/

Main Features of API Gateway

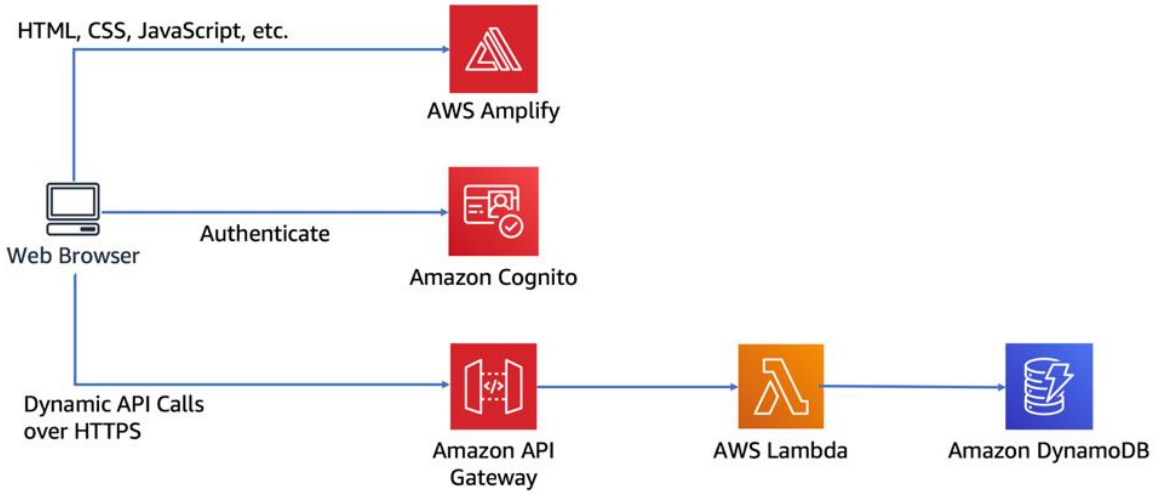
- Support for stateful (**WebSocket**) and stateless (**HTTP** and **REST**) APIs.
- **Flexible authentication** mechanisms, supports **OAuth2** and **OpenID** protocols.
- Provide **Developer portal** for publishing your APIs.
- **Canary release** deployments for safely rolling out changes.
- **CloudTrail logging** and monitoring of API usage and API changes.
- **CloudWatch** access **logging** and execution logging
- Ability to **use AWS CloudFormation** templates to enable API creation.
- Support for custom domain names. Integration with AWS X-Ray.



https://aws.amazon.com/tr/getting-started/hands-on/build-serverless-web-app-lambda-apigateway-s3-dynamodb-cognito/

Amazon API Gateway Use Cases

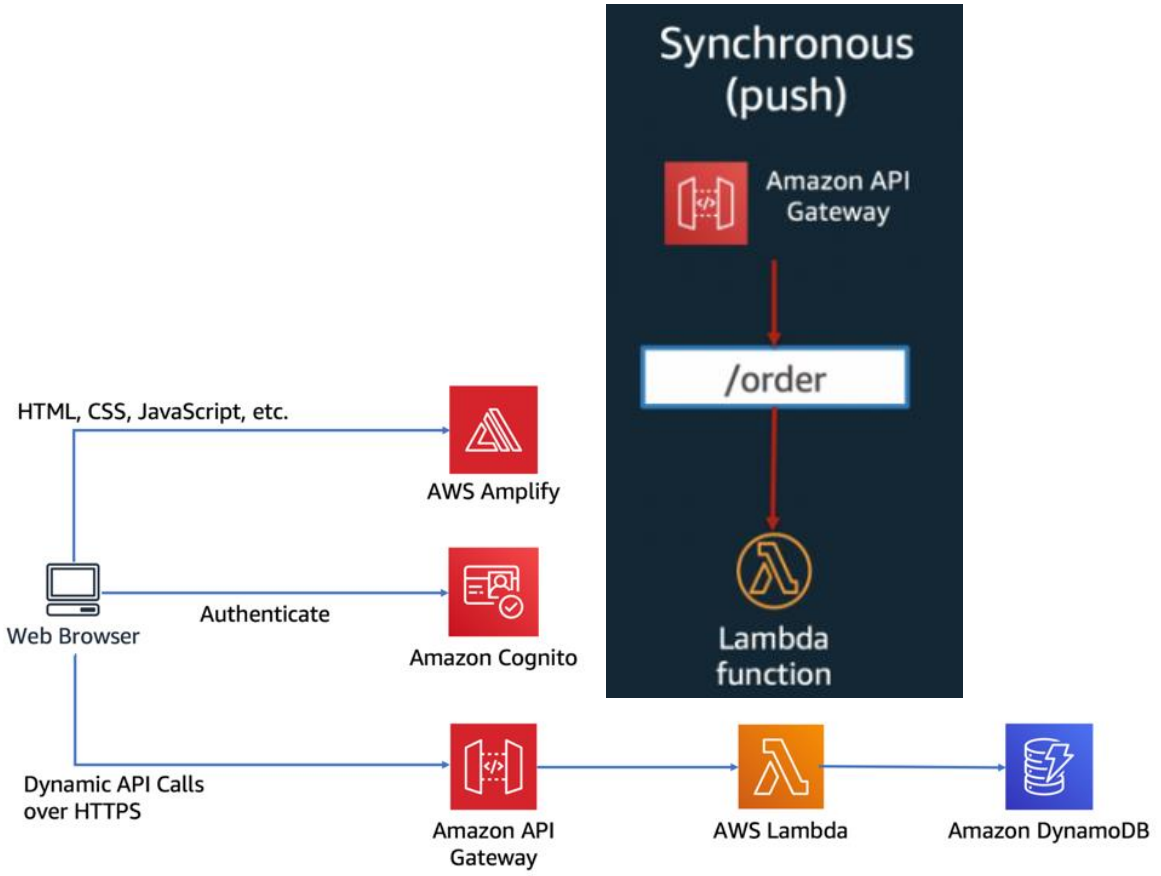
- **HTTP APIs** enable you to create **RESTful APIs** with lower latency and lower cost than REST APIs.
- Use HTTP APIs to **send requests** to AWS Lambda functions
- Create an **HTTP API** that integrates with a Lambda function on the backend.
- API Gateway **REST API** is made up of resources and methods.
  - A resource is a logical entity
  - A method corresponds to a **REST API** request
  - For example, **"/products"** could be a path to a resource
  - HTTP verbs such as **GET**, **POST**, **PUT**, **PATCH**, and **DELETE**
  - **POST /product** method can create a new product, and a **GET /basket** method can query for basket data.



https://aws.amazon.com/tr/getting-started/hands-on/build-serverless-web-app-lambda-apigateway-s3-dynamodb-cognito/

API Gateway as a Lambda Synchronous Event Sources

- **Synchronous** commands are request/response.
- **API Gateway** is a synchronous event source and provides a serverless **API proxy to Lambda**.
- Simple interaction common to create, read, update, and delete (**CRUD**) **API** actions.
- **Immediate response** to your API call.
- The disadvantage is that if something goes wrong or takes a long time, the whole **process is blocked**.
- API Gateway is **Synchronous trigger** of AWS Lambda
- Developing our Serverless **Product Microservices** CRUD operations.



https://aws.amazon.com/tr/getting-started/hands-on/build-serverless-web-app-lambda-apigateway-s3-dynamodb-cognito/

Amazon API Gateway Core Concepts

- **API Gateway**
  Creating, deploying, and managing a RESTful application programming interface (API) to expose backend HTTP endpoints. API Gateway has 3 main API types;
  - API Gateway REST API
  - API Gateway HTTP API
  - API Gateway WebSocket API
- **API Gateway REST API**
  A collection of HTTP resources and methods integrated with backend HTTP endpoints, Lambda functions.
- **API Gateway HTTP API**
  A set of paths and methods integrated with backend HTTP endpoints or Lambda functions.
- **API Gateway WebSocket API**
  A collection of WebSocket routes and route keys that are integrated with backend HTTP endpoints, Lambda functions.

Amazon API Gateway Core Concepts

- **API Deployment**
  A snapshot of your API Gateway API. The deployment must be associated with one or more API stages for clients to use it.
- **API Endpoint**
  Hostname for an API deployed to a specific Region in API Gateway. The hostname is in the form {api-id}.execute-api.{region}.amazonaws.com. 3 API endpoint types;
  - Edge-optimized API endpoint
  - Private API endpoint
  - Regional API endpoint
- **Proxy Integration**
  A simplified API Gateway integration configuration. We can set a proxy integration as HTTP proxy integration or Lambda proxy integration.
- API Gateway for HTTP proxy integration forwards all request and response between frontend and an HTTP backend. API Gateway sends the entire request as input to a backend Lambda function for Lambda proxy integration.

Amazon API Gateway -Differences between REST - HTTP API

- **HTTP API** type is **lightweight** version of Restful apis in order to be more cost efficient.
- **HTTP API** has **less feature** than **REST API**.
- **HTTP APIs** are designed for **low-latency**, **cost-effective** integrations with AWS services, including AWS Lambda, and HTTP endpoints.
- See the **Development feature** comparison of both **HTTP** and **REST APIs.**
- During the course we will use **REST APIs.**

Development	HTTP API	REST API
API caching		✓
Request parameter transformation	✓	✓
Request body transformation		✓
Request / response validation		✓
Test invocation		✓
CORS configuration	✓	✓ *
Automatic deployments	✓	
Default stage	✓	
Default route	✓	
Custom gateway responses		✓
Canary release deployment		✓

https://docs.aws.amazon.com/apigateway/latest/developerguide/http-api-vs-rest.html

Amazon API Gateway Walkthrough with AWS Management Console

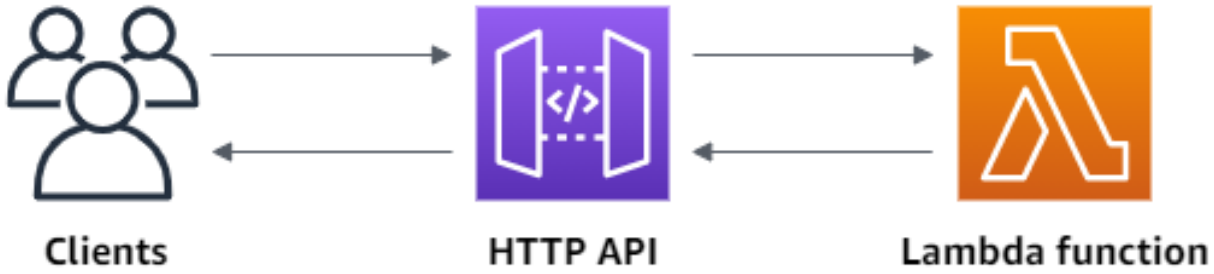
→ DEMO - Amazon API Gateway Walkthrough with AWS Management Console

Building Microservices with AWS Lambda for Synchronous APIs

→ Building Microservices with AWS Lambda for Synchronous Api-Driven Event Sources.

Build AWS Lambda Synchronous Microservices with Amazon Api Gateway

- Create a Lambda function
- Create an REST API using the API Gateway
- Invoke your API
- API Gateway routes the request to your Lambda function.
- Lambda runs the Lambda function and returns a response to API Gateway.
- API Gateway then returns a response to you.



Clean up Resources

- **Amazon CTO Werner Vogels**
  Turn off the lights before leaving the room
- If you leave from computer and shut the light off, that means you also **shut down the AWS Resources**
- Think that we are going to prepare **demo session** with **your customer**
- Follow the best practices; **clear all resources** after you finished the work at that day.
- **Don’t worry**, when you come back tomorrow, we can instantly come the same situation with the power of **AWS CDK.**

https://www.youtube.com/watch?v=8_Xs8Ik0h1w

AWS DynamoDB Serverless NoSQL Data Persistence

Building AWS DynamoDB Serverless NoSQL Data Persistence

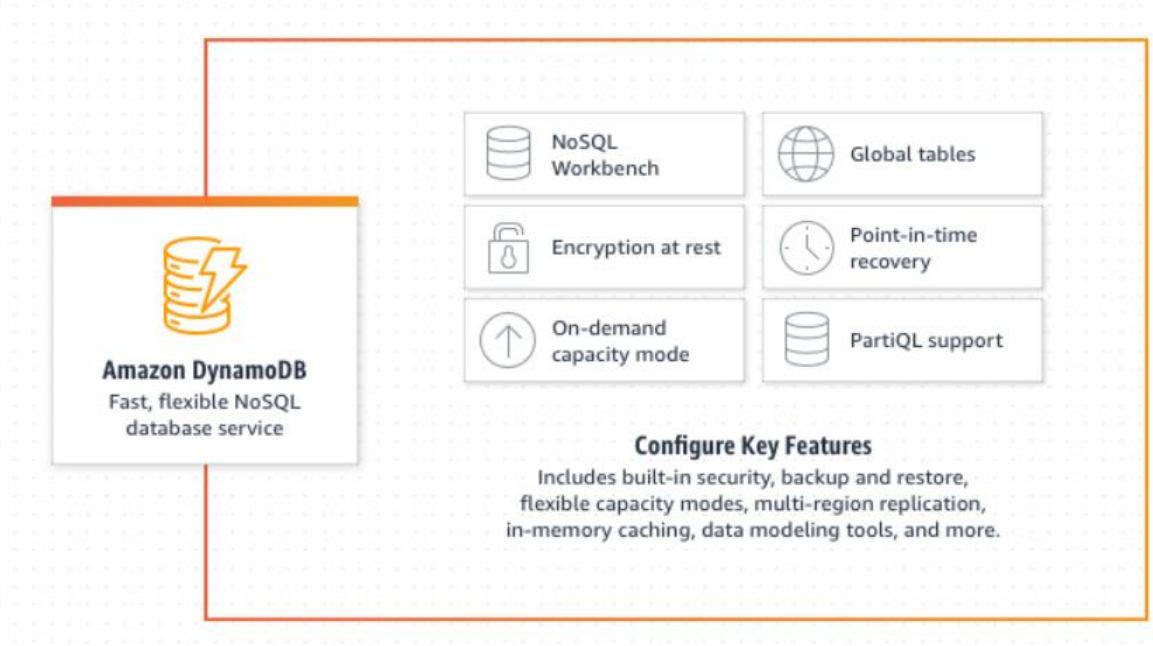
What Is Amazon DynamoDB?

- **Amazon DynamoDB** is a fully managed NoSQL database service that provides **fast** and **predictable** performance with seamless scalability.
- **Serverless**, **key-value NoSQL database** designed to run high-performance applications at any scale.
- Create **database tables** that can store and retrieve any amount of data and serve any level of request traffic.
- **Scale up or down** the throughput of your tables without downtime or performance degradation. DynamoDB provides on-demand backup capability.
- **High Availability and Durability**
  DynamoDB automatically spreads data and traffic for your tables across enough servers to meet your throughput.

https://aws.amazon.com/dynamodb/

AWS DynamoDB Core Concepts - Tables, Items, Attributes, Indexes

- Tables, Items, and Attributes are the core components.
- Uses primary keys and secondary indexes to uniquely identify each item in a table for greater query flexibility.
- **Tables**
  DynamoDB stores data in tables. A table is a collection of data items. For example, see the sample table People.
- **Items**
  Each table contains zero or more items. An item is a set of attributes that can be uniquely identified among all of the other items.
- **Attributes**
  Each item is composed of one or more attributes. An attribute is a fundamental data element.



https://aws.amazon.com/dynamodb/

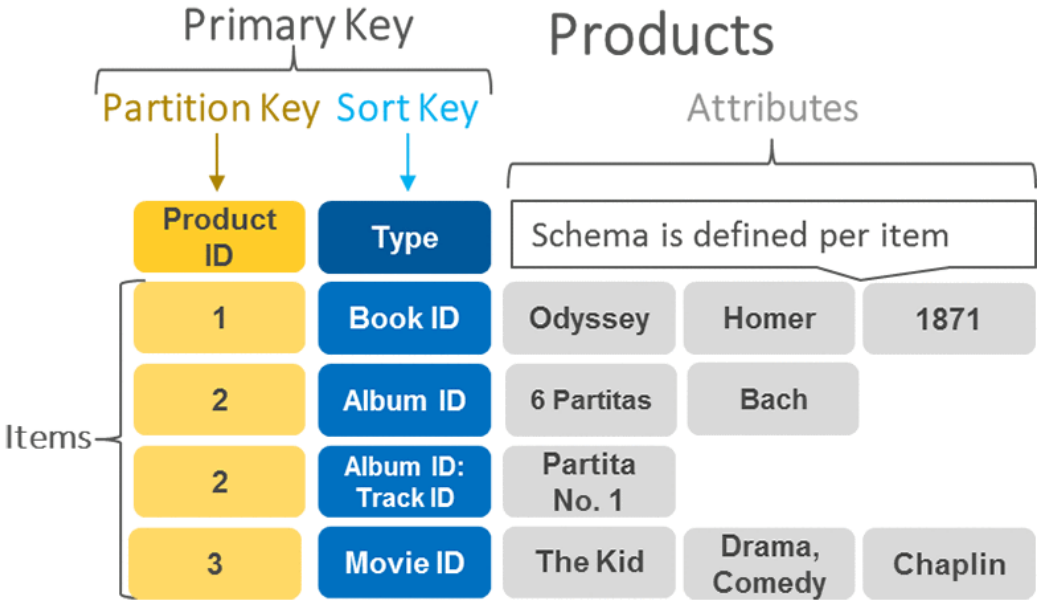
AWS DynamoDB Core Concepts - Tables, Items, Attributes, Indexes

People

https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/HowItWorks.CoreComponents.html

DynamoDB Primary Key, Partition Key and Sort Key

- A primary key **uniquely identifies** each item in the table, so no two items can have the same key. DynamoDB supports two different kinds of primary keys:
  - Partition key
  - Partition key and sort key
- **Partition key**
  A simple primary key, composed of one attribute known as the partition key.
- **Partition key and Sort Key**
  It is Referred to as a composite primary key, this type of key is composed of two attributes. The first attribute is the partition key, and the second attribute is the sort key.
- DynamoDB uses the partition key value as input to an internal hash function. A composite primary key gives you additional flexibility when **querying data**.



https://aws.amazon.com/blogs/database/choosing-the-right-dynamodb-partition-key/

AWS DynamoDB Walkthrough with AWS Management Console

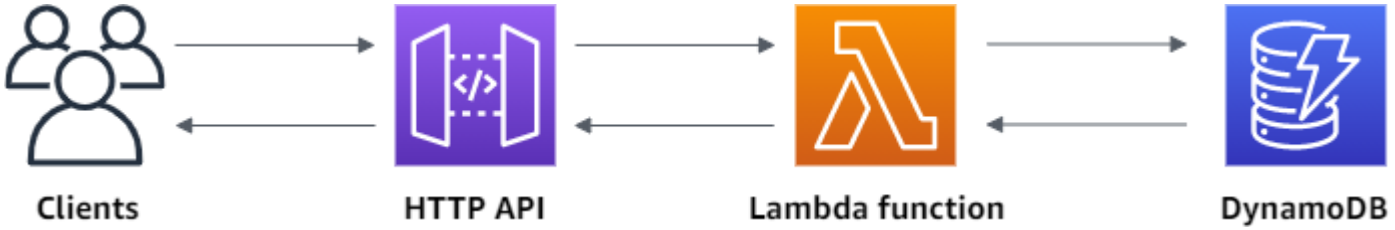
→ DEMO - AWS DynamoDB Walkthrough with AWS Management Console

Building RESTful Microservices with AWS Lambda, Api Gateway and DynamoDb

Building RESTful Microservices with AWS Lambda, Api Gateway and DynamoDb

RESTful Microservices with AWS Lambda, Api Gateway and DynamoDb

- Create a Serverless API that creates, reads, updates, and deletes items from a **DynamoDB table**.
- Create a **DynamoDB table** using the DynamoDB console.
- create a **Lambda function** using the AWS Lambda console.
- Create an **REST API** using the API Gateway console. Lastly, we test your API.



- Clients **send request** our **microservices** by making HTTP API calls.
- Amazon **API Gateway** hosts RESTful HTTP requests and responses to customers.
- AWS **Lambda** contains the business logic to process **incoming API calls** and leverage DynamoDB as a persistent storage.
- Amazon DynamoDB **persistently** stores **microservices** data and scales based on demand.

Clean up Resources

- **Amazon CTO Werner Vogels**
  Turn off the lights before leaving the room
- If you leave from computer and shut the light off, that means you also **shut down the AWS Resources**
- Think that we are going to prepare **demo session** with **your customer**
- Follow the best practices; **clear all resources** after you finished the work at that day.
- **Don’t worry**, when you come back tomorrow, we can instantly come the same situation with the power of **AWS CDK.**

https://www.youtube.com/watch?v=8_Xs8Ik0h1w

AWS Serverless Deployments IaC with AWS CDK

→ Learn AWS Serverless Deployments IaC with AWS CDK (Cloud Development Kit)

Serverless Deployments IaC with AWS CDK

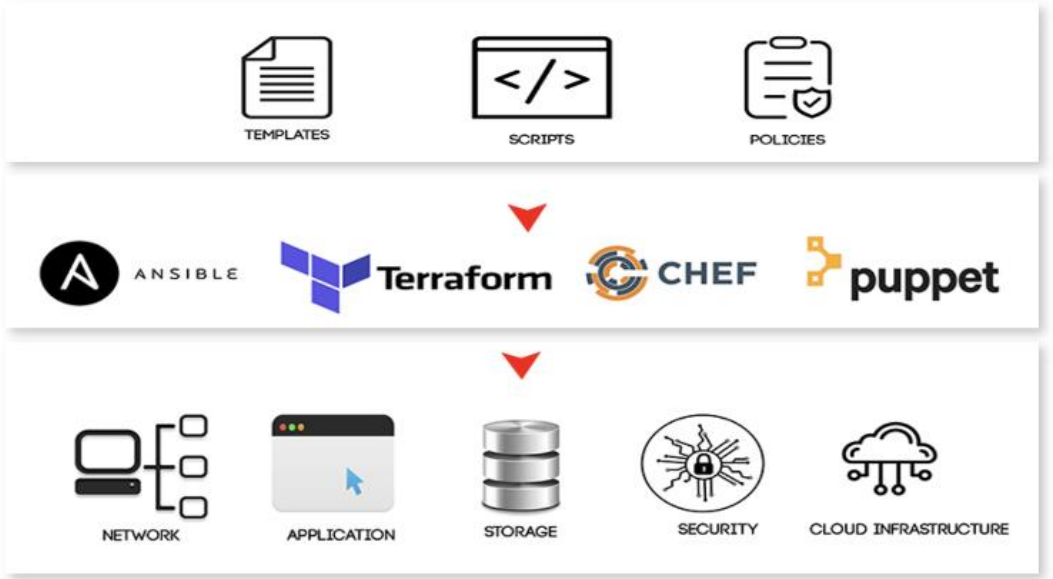
- What is IaC - Infrastructure as Code
- AWS Cloud Formation
- AWS CDK - Cloud Development Kit
- AWS CDK - Overview - Features - Core Concepts
- Importance of CDK from Werner Vogels the CTO of AWS
- AWS CDK - Application - Stacks - Resources via Constructs => CF
- AWS CDK Construct Levels L1-L2-L3 Constructs
- AWS CDK CDK Solution Constructs and CDK Serverless Patterns and so on..

aws

Cloud Development Kit

What is IaC - Infrastructure as Code

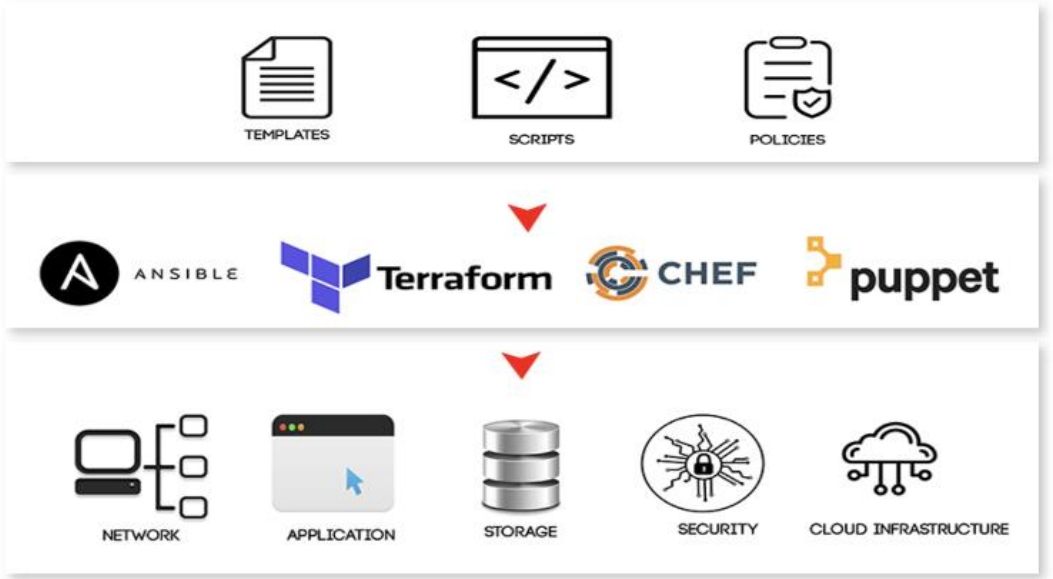
- **Infrastructure as Code** or IaC is the process of provisioning and managing infrastructure defined through code
- It allows users to easily **edit** and **distribute configurations**, you can create **reproducible** infrastructure configurations.
- IaC is a process that **automates** the **provisioning** and management of cloud resources.
- IaC software takes input **scripts describing** the **desired state** and then communicates with the cloud vendors.
- **Programmable Infrastructure**; IaC configures infrastructure exactly like programming software.
- IaC is one way of **raising the standard** of infrastructure management and time to deployment.
- IaC can safely create and configure infrastructure elements **in seconds**.



https://dzone.com/articles/5-principles-of-infrastructure-as-code-iac

Benefits of Infrastructure as Code

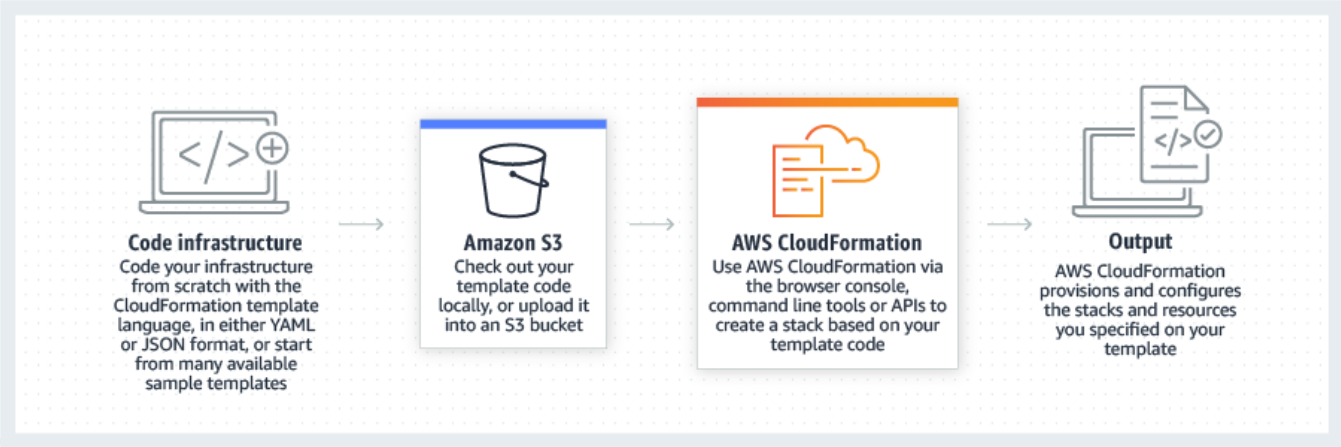
- **Speed**
  By avoiding manual intervention, infrastructure deployments are quick and safe.
- **Consistency**
  Deploy identical infrastructure across the board, avoiding edge-cases and one-off configurations.
- **Reusability**
  IaC makes it easy to create reusable modules.
- **Reduced cost**
  IaC allows virtual machines to be managed programmatically
- **Source control**
  Code can be checked in source control for increased transparency and accountability.
- **Agility**
  DevOps has made software delivery more efficient.



https://dzone.com/articles/5-principles-of-infrastructure-as-code-iac

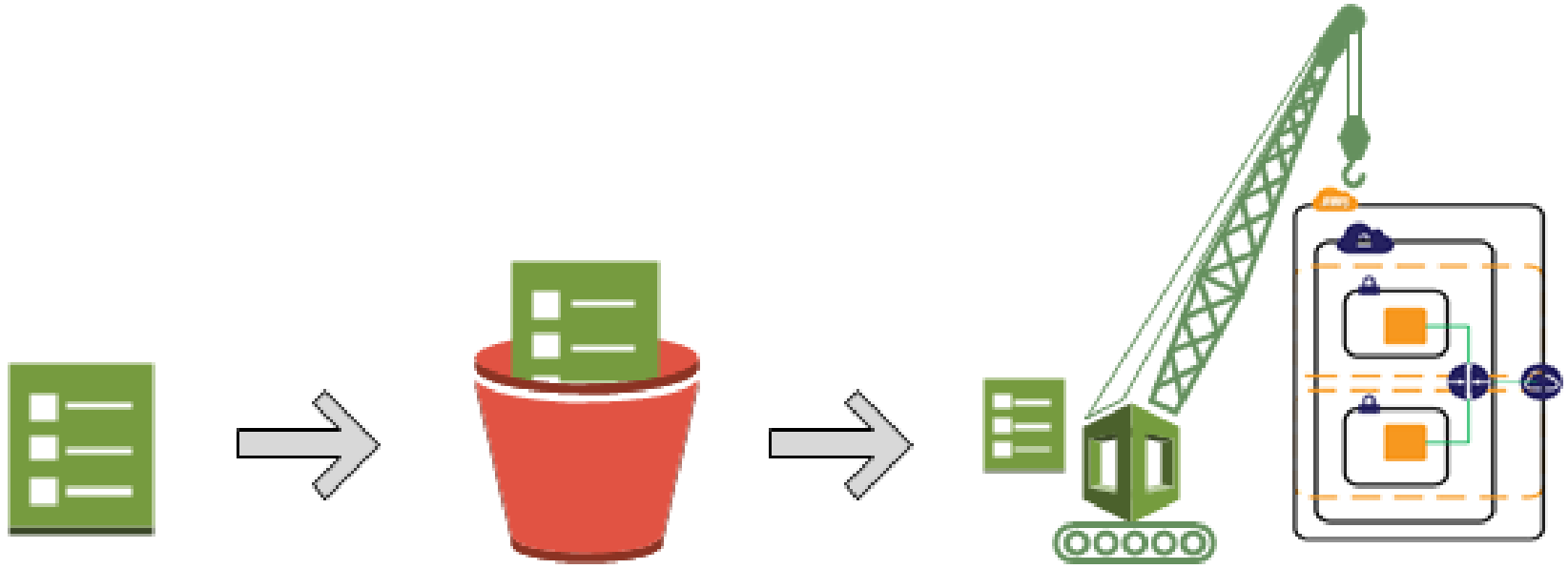
AWS Cloud Formation

- **Model**, **provision**, and **manage** AWS resources by treating **infrastructure as code.**
- Infrastructure **automation platform** for AWS that deploys AWS resources in a **repeatable**, **testable** and **auditable** manner.
- Uses **template files** to automate the setup of AWS resources
- Enables you to **create** and **provision** AWS infrastructure deployments **predictably** and **repeatedly**.
- Described as **infrastructure automation** or Infrastructure-as-Code (IaC) tool.
- Create a **template** that describes all the AWS resources and **CloudFormation** takes care of **provisioning**.



https://aws.amazon.com/cloudformation/

How does AWS CloudFormation work?



1 Create or use an existing template

2 Save locally or in S3 bucket

3 Use AWS CloudFormation to create a stack based on your template. It constructs and configures your stack resources.

```
{
  "AWSTemplateFormatVersion" : "2010-09-09",
  "Description" : "A simple EC2 instance",
  "Resources" : {
    "MyEC2Instance" : {
      "Type" : "AWS::EC2::Instance",
      "Properties" : {
        "ImageId" : "ami-0ff8a91507f77f867",
        "InstanceType" : "t2.micro"
      }
    }
  }
}
```

https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/cfn-whatis-howdoesitwork.html

What is AWS Cloud Development Kit (AWS CDK) ?

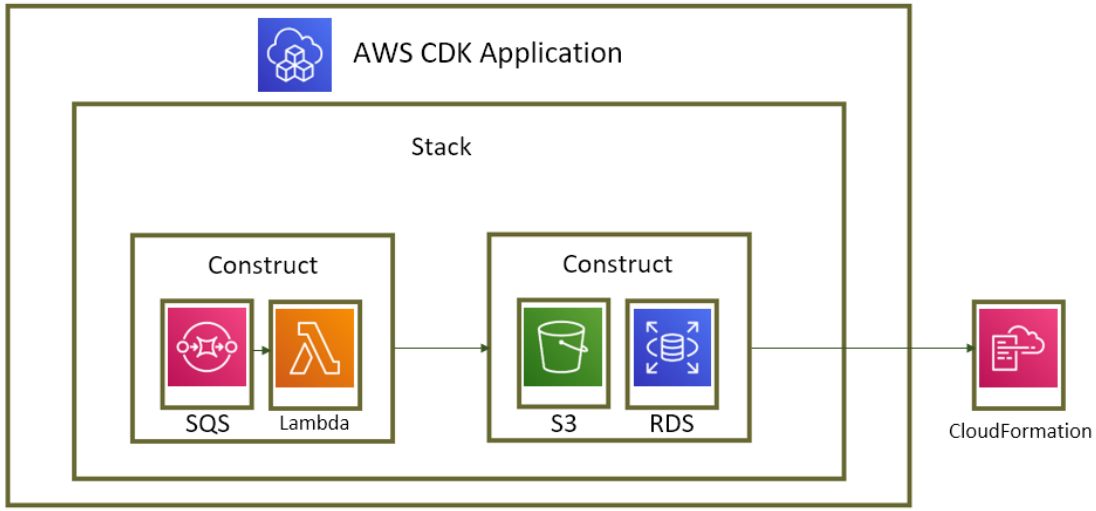
- **Open-source** software development framework to **define** your cloud application **resources** using **familiar programming languages**.
- Uses the **familiarity** and **expressive** power of **programming languages** for modeling your applications.
- Provides **high-level components** called constructs that **preconfigure cloud resources** with proven defaults.
- Provides a **library of constructs** in many programming languages to easily automate **AWS infrastructure**.
- **Provisions** your **resources** in a safe, repeatable manner through **AWS CloudFormation**.
- First-class support for **TypeScript**, **JavaScript**, **Python**, **Java**, and **C#.**
- **Why** we use AWS CDK ?

AWS CDK: What is it?

Benefits of AWS CDK

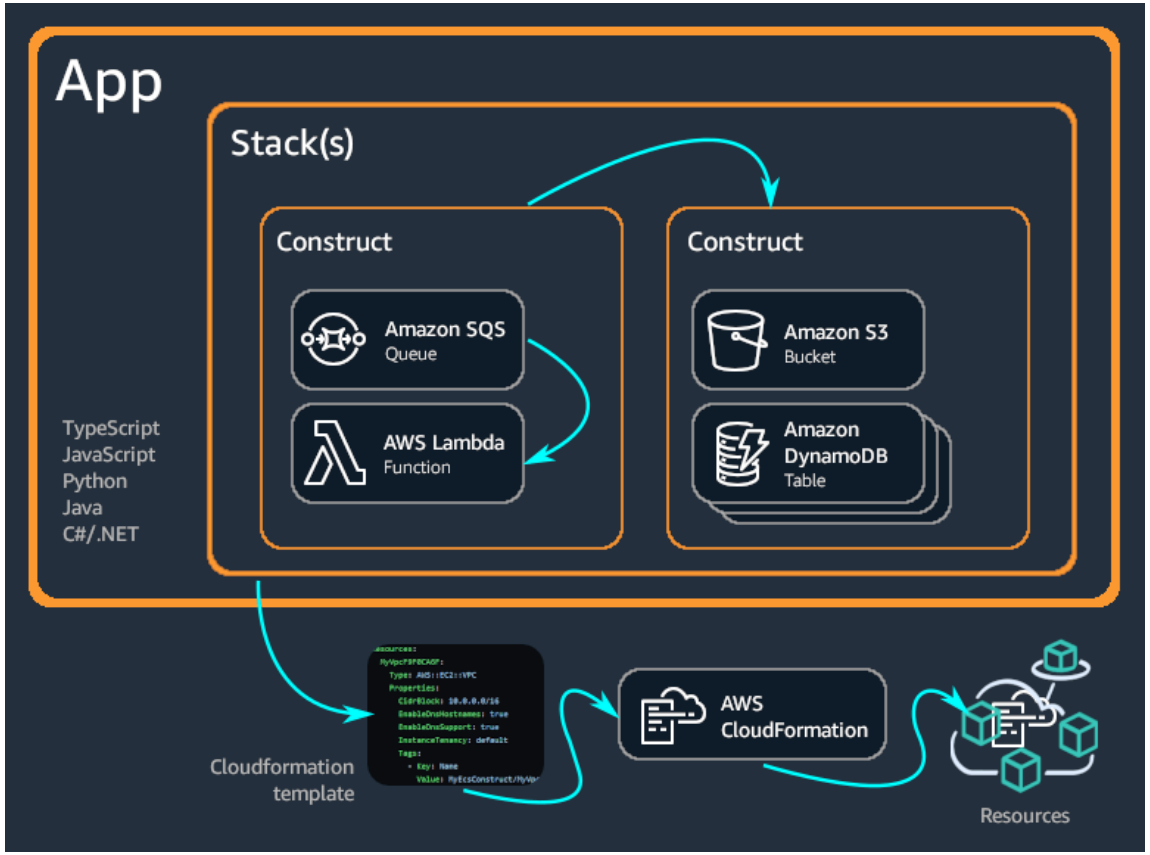
- **Choose a programming language of your choice**
  AWS CDK currently supports JavaScript, TypeScript, Python, Java, C#, and Go programming languages..
- **Powered by AWS CloudFormation**
  Enables you to define your infrastructure with code and provision it through AWS CloudFormation.
- **Auto-complete and inline documentation**
  Autocomplete and inline documentation while coding the infrastructure.
- **Deploy infrastructure and runtime code together**
  Reference your runtime code assets in the same project with the same programming language.
- **Developer-friendly command-line interface (CLI)**
  Enables you to interact with your CDK applications.

aws Cloud Development Kit



AWS CDK Core Concepts - Apps - Stacks - Constructs - Environments

- **Apps**
  Include everything needed to deploy your app to a cloud environment.
- **Stack**
  The unit of deployment in the AWS CDK is called a stack.
- **Constructs**
  The basic building blocks of AWS CDK apps. A construct represents a “cloud component”.
- **Environments**
  Each Stack instance in your AWS CDK app is explicitly or implicitly associated with an environment.



https://docs.aws.amazon.com/cdk/v2/guide/home.html

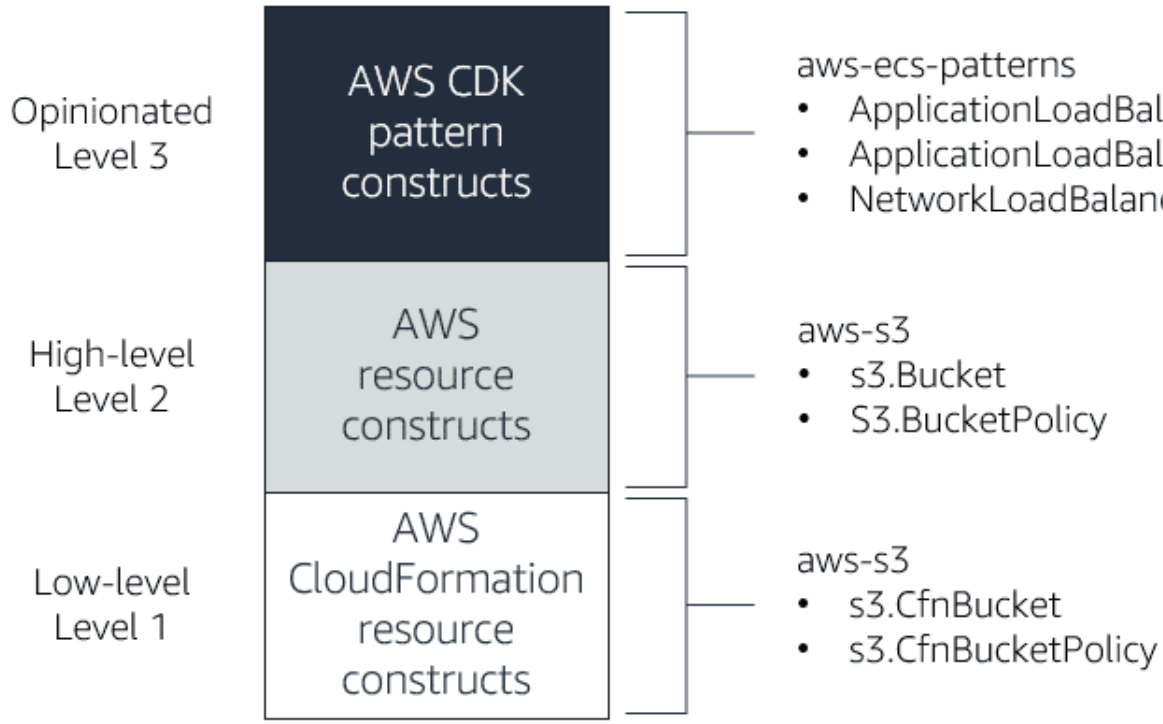
Example CDK application includes Apps, Stacks, Constructs and Environments

```
import { App, Stack, StackProps } from 'aws-cdk-lib';
import * as s3 from 'aws-cdk-lib/aws-s3';

class HelloCdkStack extends Stack {
  constructor(scope: App, id: string, props?: StackProps) {
    super(scope, id, props);

    new s3.Bucket(this, 'MyFirstBucket', {
      versioned: true
    });
  }
}

const app = new App();
new HelloCdkStack(app, "HelloCdkStack");
```

https://docs.aws.amazon.com/cdk/v2/guide/constructs.html

AWS Solutions Constructs with Levels L1 - L2 - L3 Pattern Construct

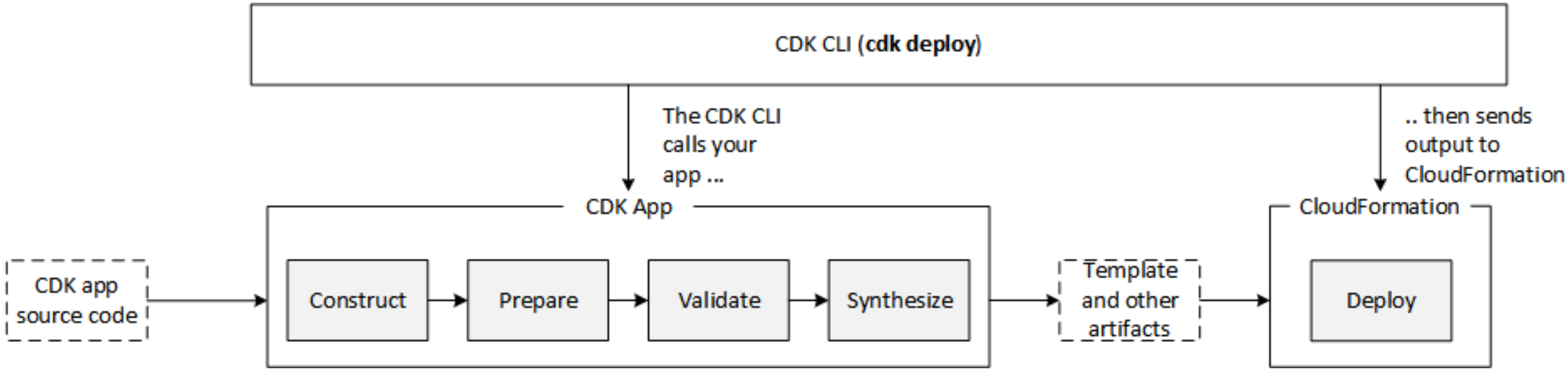
- **Level 1 (L1) construct**
  Direct representations of CloudFormation resources. Must provide all the required CloudFormation attributes. They are named CfnXyz.
- **Level 2 (L2) construct**
  Represents a particular cloud resource. S3 bucket. You don't have to configure every configuration attribute. higher-level, intent-based API. s3.Bucket class represents an Amazon S3, such as bucket.addLifeCycleRule()
- **Level 3 (L3) construct = Pattern Construct**
  Represents a bunch of cloud resources that work together to accomplish a particular task. You can create an ApplicationLoadBalancedFarageteService. Higher-level constructs, which we call patterns.



https://docs.aws.amazon.com/cdk/v2/guide/home.html

AWS CDK Lifecycle

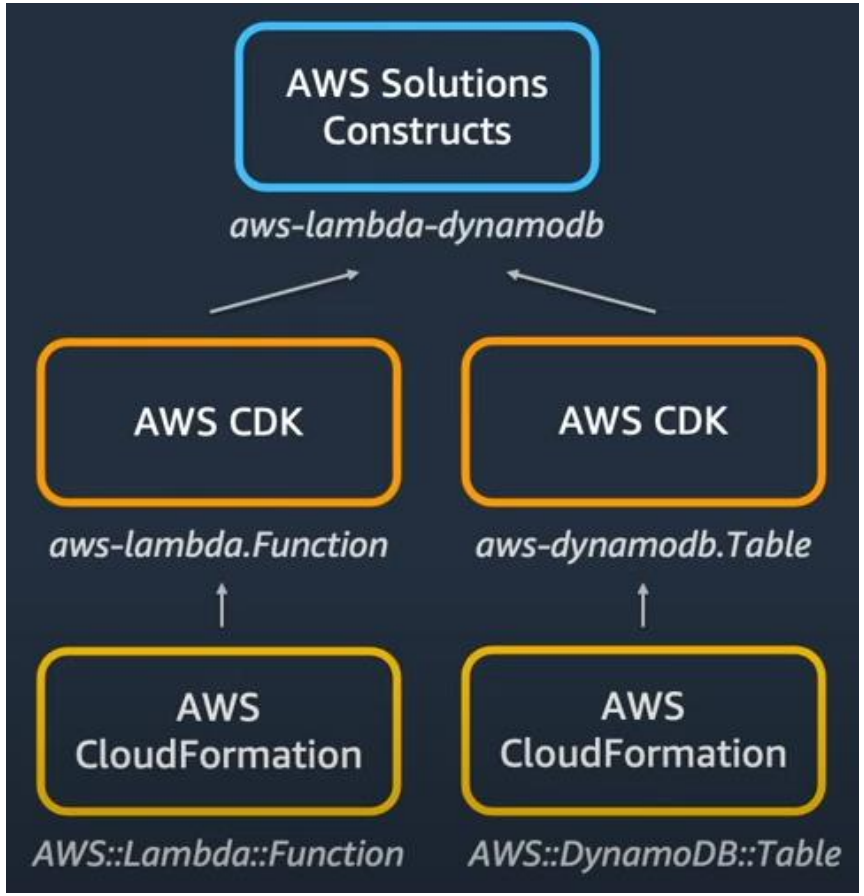
- **Construction – Preparation – Validation – Synthesis - Deployment**



https://docs.aws.amazon.com/cdk/v2/guide/apps.html

CDK Serverless Patterns with Solution Constructs

- **CDK Pattern**
  https://serverlessland.com/patterns?framework=CDK
- **CDK Serverless Patterns**
  https://github.com/cdk-patterns/serverless
- **Construct Hub**
  https://constructs.dev/
- **The Big Fan Pattern**
  https://github.com/cdk-patterns/serverless/tree/main/the-big-fan



https://www.youtube.com/watch?v=9YW1sL0dFV8

Serverless + CDK Automation + Integration Patterns = AWSome!

- **AWS Serverless** will be our application development framework.
- **AWS CDK** is IaC tool that we will develop whole infrastructure with typescript coding
- **Integration Patterns** with Queue-Chaning, Publish-Subscribe and Fan-out design patterns
- **AWSome** Microservices !

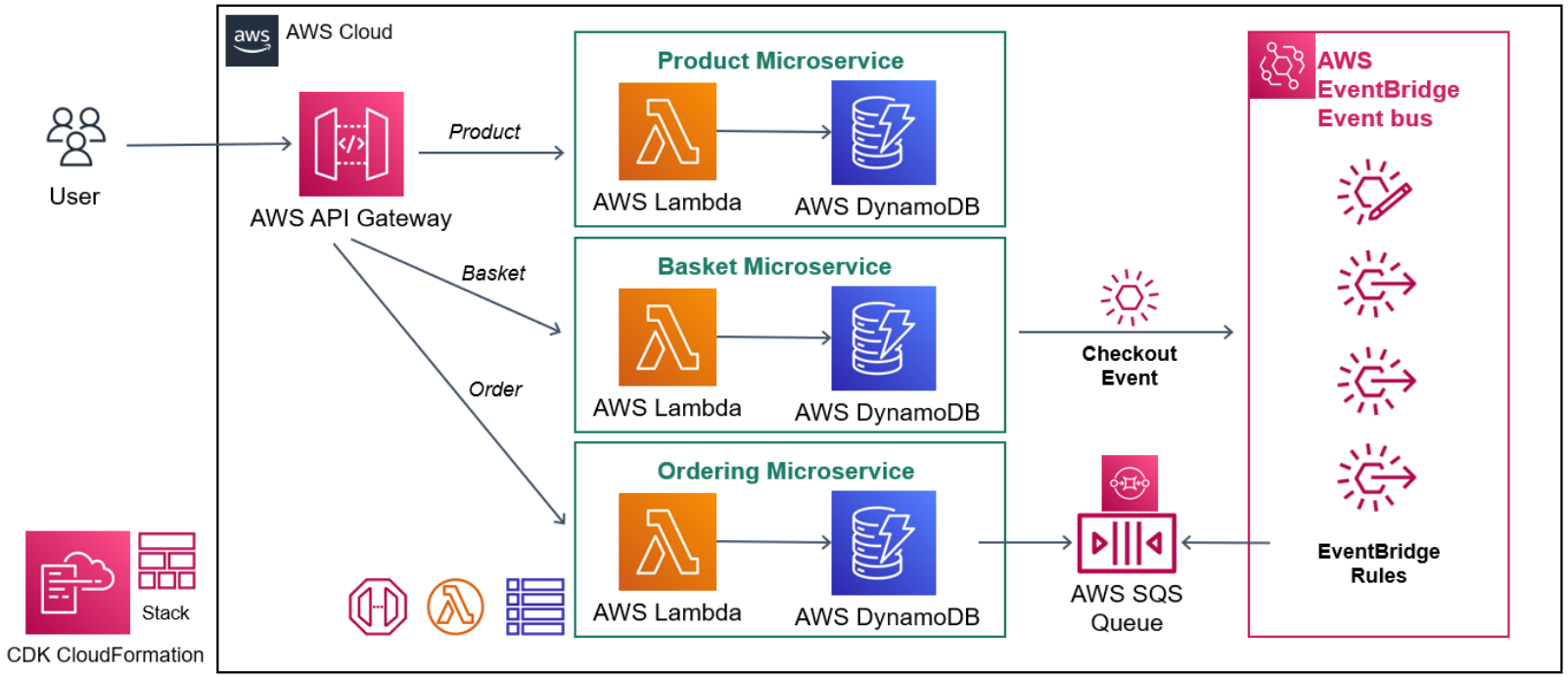
https://www.youtube.com/watch?v=ttJAIQf7cTw

Importance of CDK for Werner Vogels the CTO of AWS

AWS re:Invent 2021 - Keynote with Dr. Werner Vogels
https://www.youtube.com/watch?v=8_Xs8Ik0h1w

AWS CDK - Hands-on IaC Development

- Learned about AWS CDK
- Start our project development with IaC using AWS CDK
- Rest of the course, we will develop our e-commerce application with IaC using AWS CDK.
- Download some tools so have to download Prerequisites.
- Prerequisites and Tools for developing Serverless Applications on AWS.

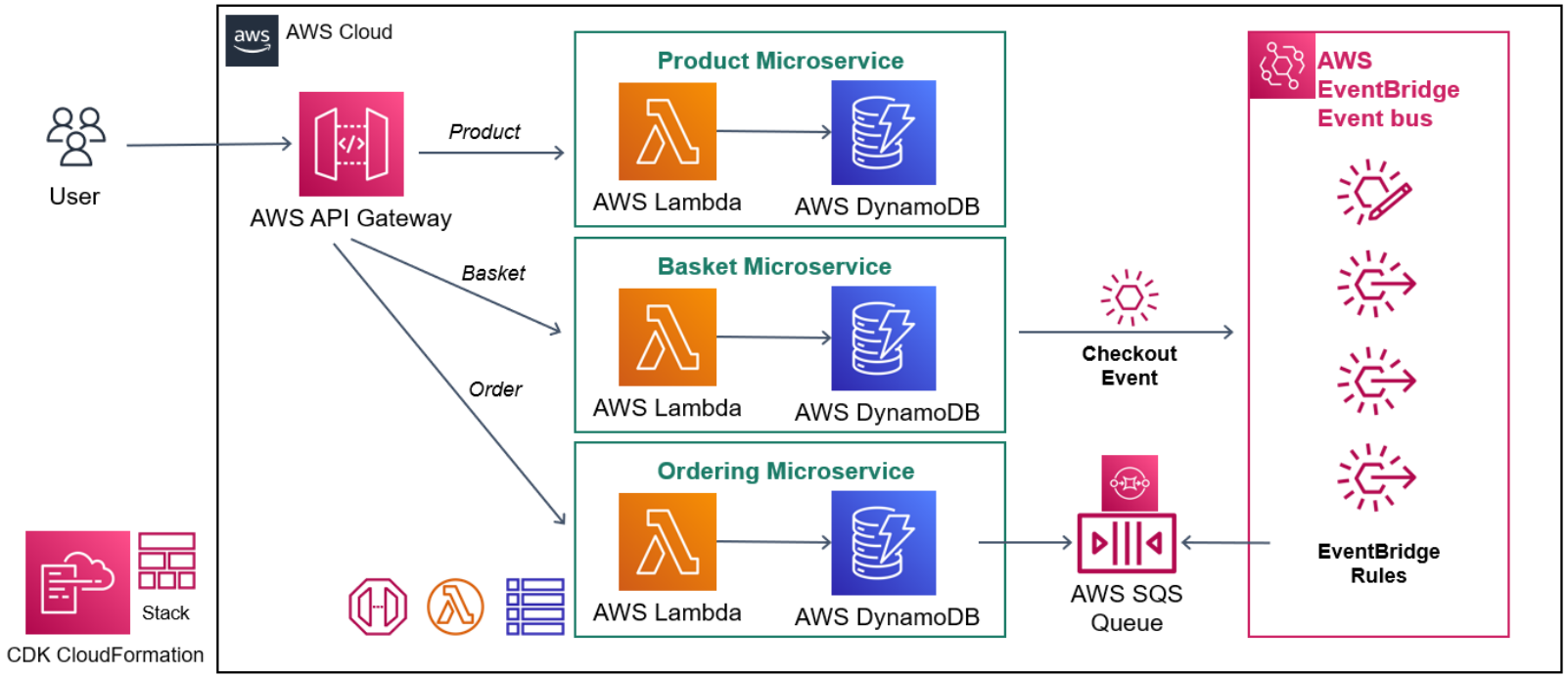


Prerequisites and Tools for developing Serverless Applications on AWS

Prerequisites and Tools for developing Serverless Applications on AWS

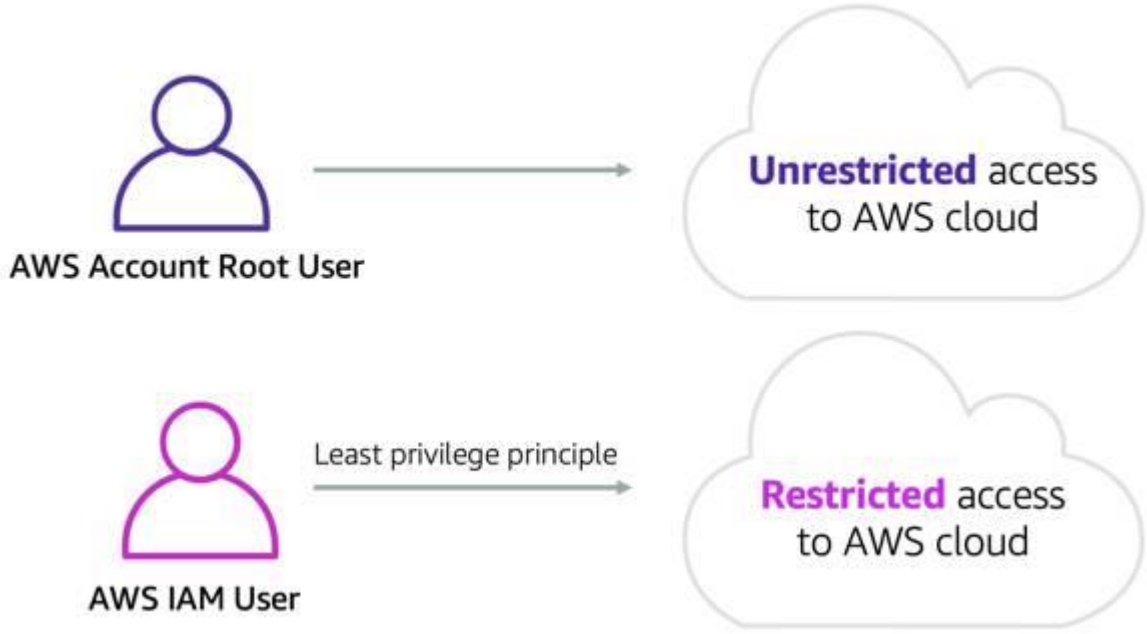
Prerequisites and Tools

- **5 main** Prerequisites;
- AWS Account and User
- AWS CLI
- Node.js
- AWS CDK Toolkit
- IDE for your programming language = Visual Studio Code



Prerequisites 1 - AWS Account and User

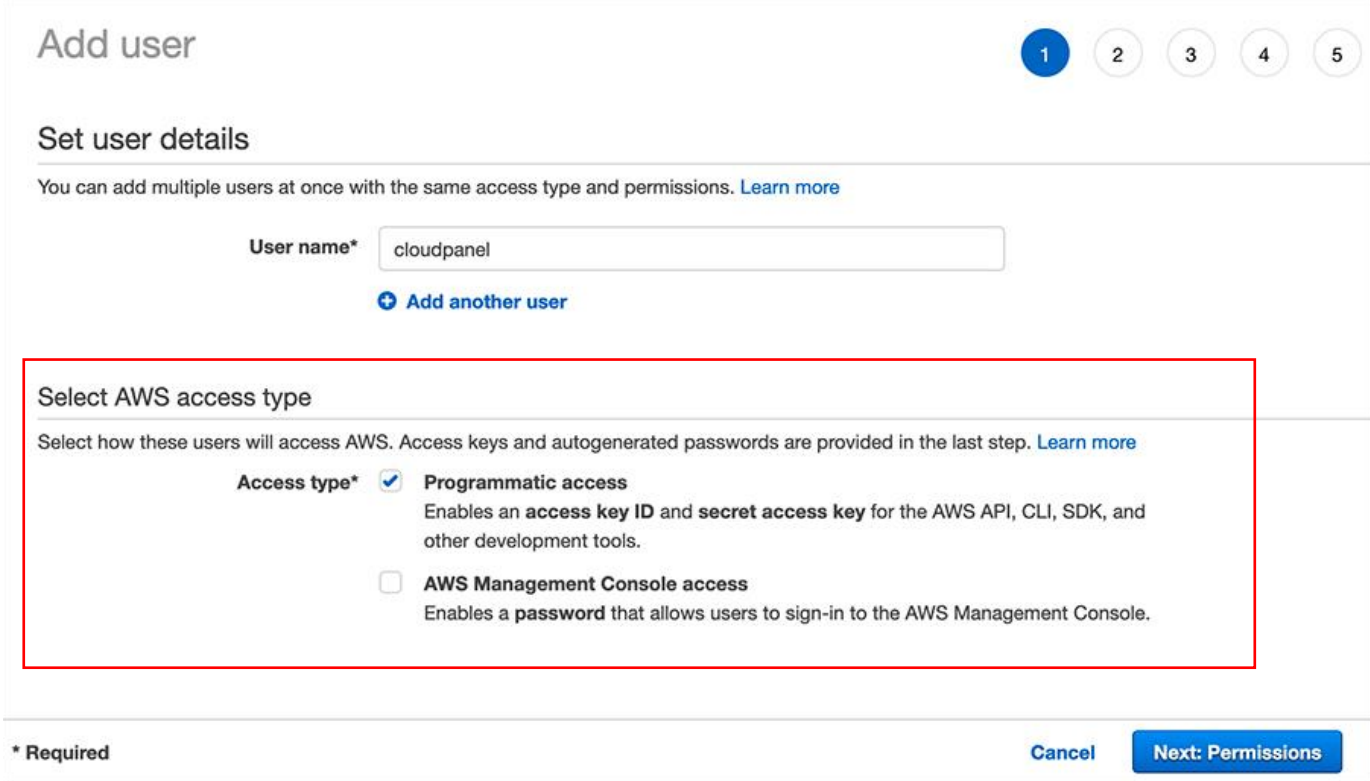
- Create **user-specific AWS account** under root user account that they have own login and passwords
- Define **Programmatic and Console Access**
- **Programmatic access is required** for our course, because we will use all interactions with AWS resources like **AWS Console, AWS CLI, AWS CDK and AWS SDK.**
- Follow me from AWS Console or follow article below;
- Create IAM User Account and Configure for Programmatic and Console Access
  - **Don’t forget to allow Programmatic Access**



https://trailhead.salesforce.com/en/content/learn/modules/aws-identity-and-access-management/set-iam-policies

Prerequisites 1 - AWS Account and User

- When creating a user, we have an option about **AWS Access Types**
  - **Programmatic Access**
  - **AWS Management Console Access**
- **Programmatic Access**
  Enables **access key ID** and **secret access key** for the **AWS API, CLI, SDK,** and other development tools.
- During the course we will almost use all of these programmatic access types and of course use AWS Console every time



Prerequisites 2 - AWS CLI

- The **AWS Command Line Interface (CLI)** is a unified tool to manage your AWS services.
- Open this official link
- Installing or updating the latest version of the AWS CLI
- Make sure that you are in v2
- Quick Setup of AWS CLI
- "**aws configure**" command is the fastest way to set up your AWS CLI installation.
- Access key ID
- Secret access key
- AWS Region
- Output format

Prerequisites 3 - NodeJS

- **Node.js** is an **open-source, cross-platform, back-end JavaScript runtime environment** that runs on the V8 engine and executes JavaScript code outside a web browser.
- Develop our **lambda microservices** with nodejs runtime.
- **CDK** use npm - node package management and developing with typescript.
- Installing or updating the latest version of the Node JS
- AWS CDK uses Node.js (>= 10.13.0, except for versions 13.0.0 - 13.6.0).
- Download **LTS**

Prerequisites 4 - AWS CDK Toolkit

- The AWS CDK Toolkit, the CLI command cdk, is the primary tool for **interacting with your AWS CDK** app.
- Executes your app, manages the application model you defined, and produces and deploys the **AWS CloudFormation** templates generated by the AWS CDK.
- **AWS CDK Toolkit** is installed with the npm - Node Package Manager.
- **npm install -g aws-cdk**
- Install latest version

https://aws.amazon.com/blogs/developer/tag/aws-cdk/

Prerequisites 5 - Visual Studio Code

- **Visual Studio Code** is a code editor redefined and optimized for building and debugging modern web and cloud applications.
- **IDE** for our cdk typescript programming language.
- Installing or updating the latest version of the VS Code
- Successfuly installed Visual Studio Code.
- Ready for the our first **AWS CDK Application**.

https://code.visualstudio.com/

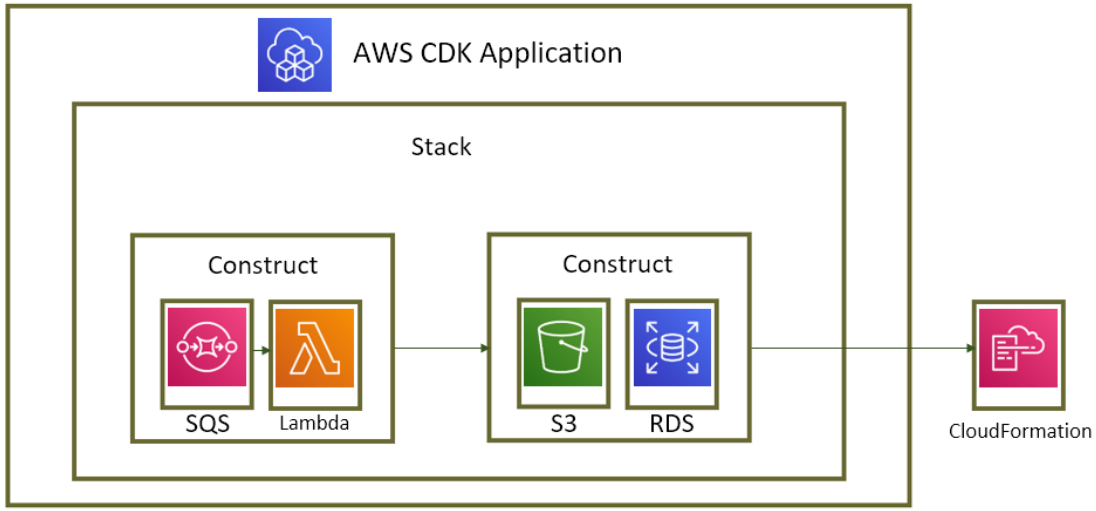
Getting Started with AWS CDK with Developing our first CDK Application

→ Getting Started with AWS CDK with Developing our first CDK Application.

Getting Started with AWS CDK

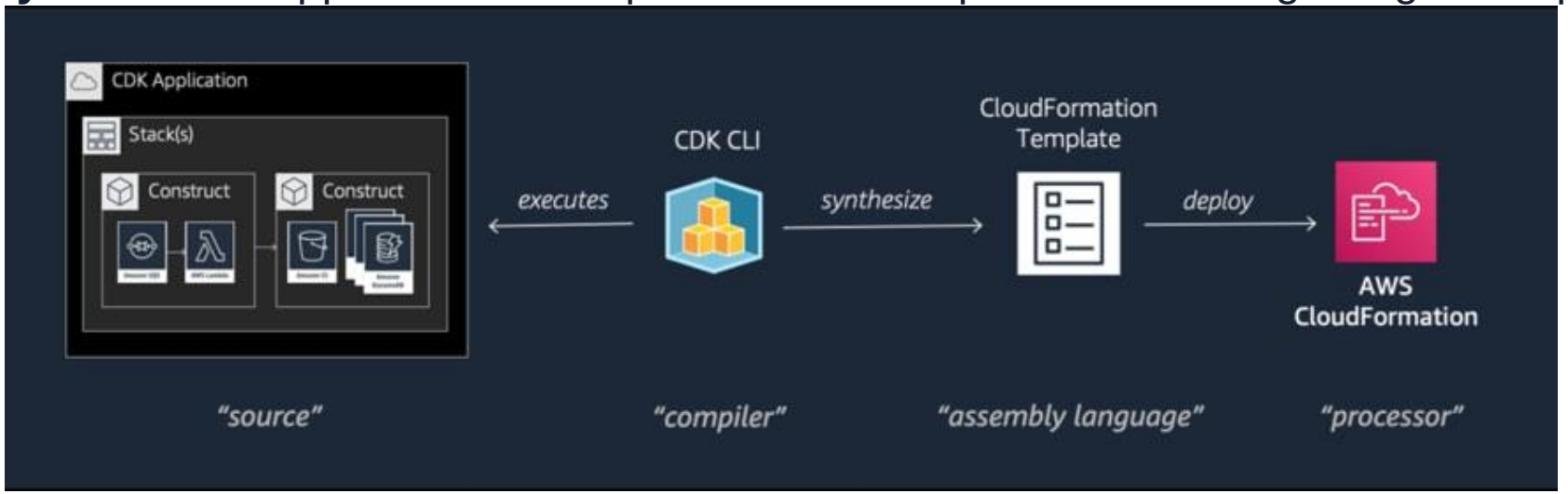
- Learn **structure** of a **AWS CDK project**.
- Learn how to use the **AWS Construct Library** to define AWS resources using code.
- Learn how to **synthesize**, **diff**, and **deploy** collections of resources using the AWS CDK Toolkit command-line tool.
- Breakdown what are going to do;
  - Create the app from a **template** provided by the AWS CDK
  - Add code to the app to create resources within stacks
  - **Build** the app its optional; the **AWS CDK Toolkit** will do it for you
  - **Synthesize** stacks in the app to create an AWS CloudFormation template
  - **Deploy** one or more stacks to your AWS account that we configured.
- Follow best practices when developing cdk applications;
  - The **build** step **catches syntax** and type errors.
  - The **synthesis** step catches logical errors in defining your AWS resources
  - The **deployment** may find permission issues.

aws Cloud Development Kit



Bootstrapping CDK Stack

- **Lifecycle** of CDK applications. It required to bootstrap CDK at the beginning of the project.



- In official explanation, **Deploying AWS CDK** apps into an AWS environment (a combination of an AWS account and region) may require that you provision resources the AWS CDK needs to perform the deployment.
- These resources include an **Amazon S3 bucket** for storing files and **IAM roles** that grant permissions needed to perform deployments.

Clean up Resources

- **Amazon CTO Werner Vogels**
  Turn off the lights before leaving the room
- If you leave from computer and shut the light off, that means you also **shut down the AWS Resources**
- Think that we are going to prepare **demo session** with **your customer**
- Follow the best practices; **clear all resources** after you finished the work at that day.
- **Don’t worry**, when you come back tomorrow, we can instantly come the same situation with the power of **AWS CDK.**

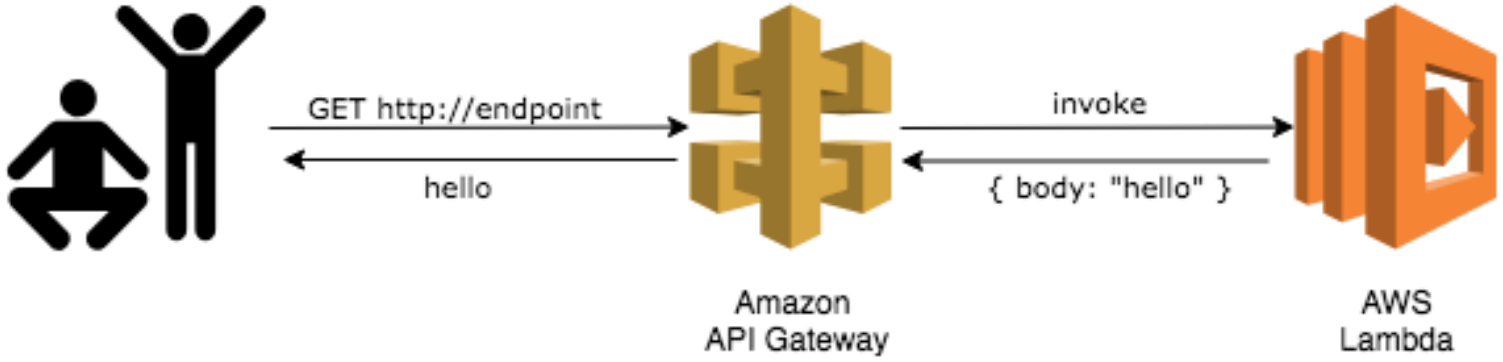
https://www.youtube.com/watch?v=8_Xs8Ik0h1w

Api Gateway-Lambda Synchronous RESTful Microservices with CDK

Developing Api Gateway-Lambda Synchronous RESTful Microservices with CDK.

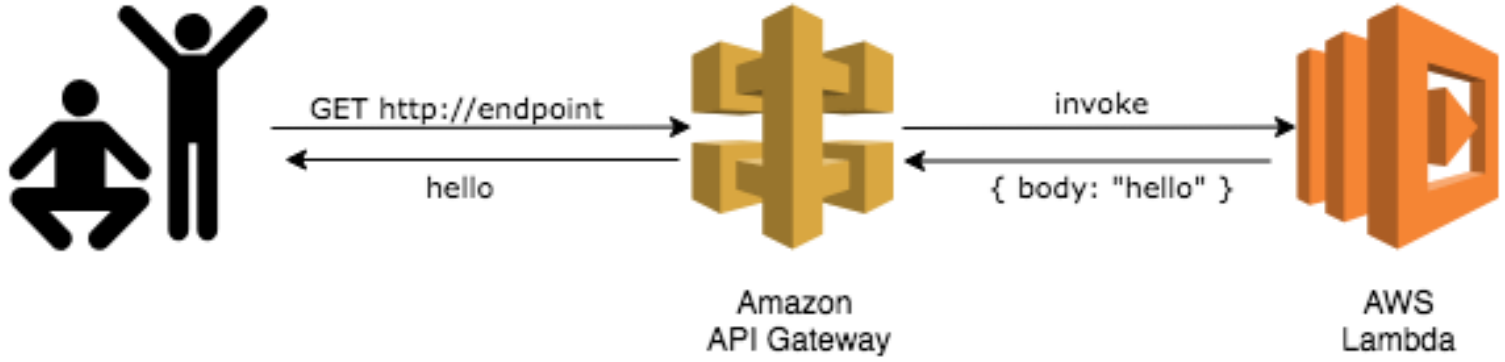
Synchronous RESTful Microservices with CDK

- Add a Lambda function with an API Gateway endpoint in front of it.
- Users will be able to hit the URL in the endpoint
- Receive a hello greeting from our function.



Synchronous RESTful Microservices with CDK

- These are 2 development areas into our projects.
- **Developing CDK** Infasturcture
- **Developing Lambda Microservices** Function itself



Clean up Resources

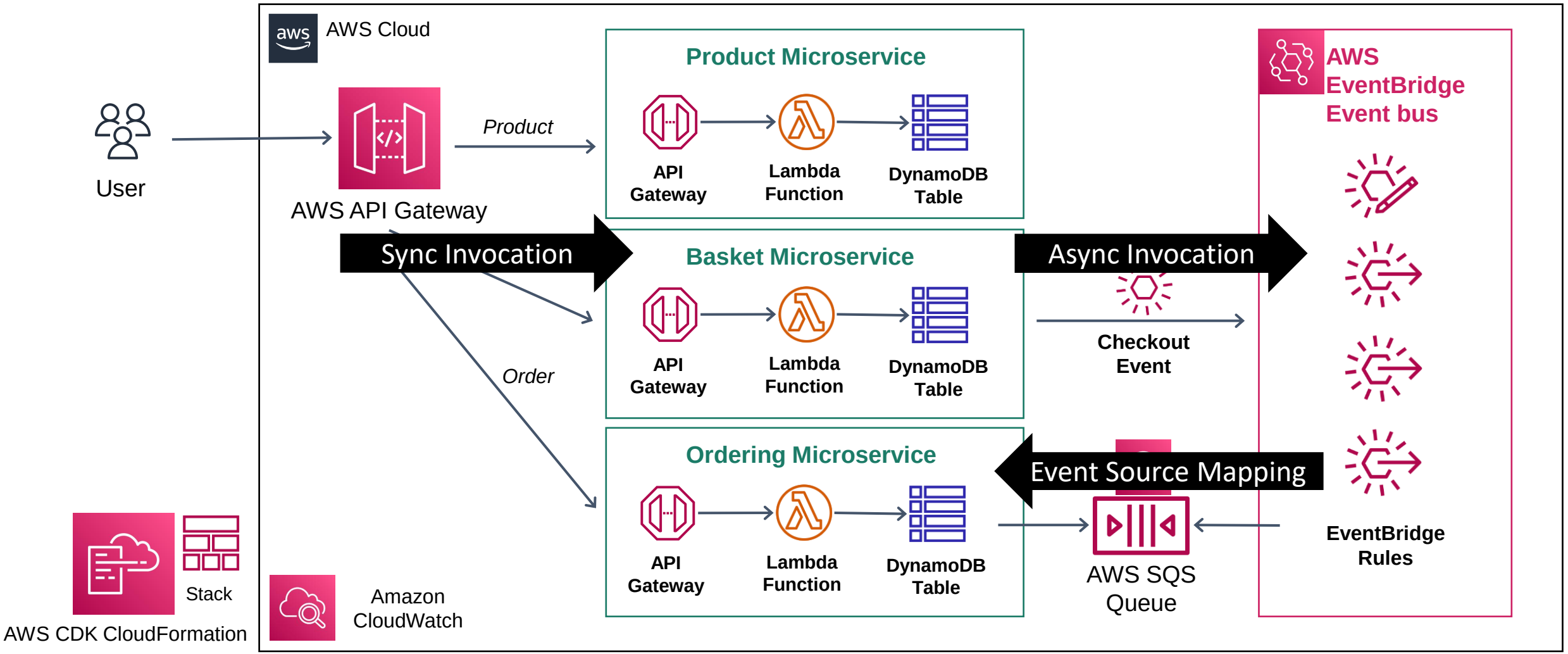
- **Amazon CTO Werner Vogels**
  Turn off the lights before leaving the room
- If you leave from computer and shut the light off, that means you also **shut down the AWS Resources**
- Think that we are going to prepare **demo session** with **your customer**
- Follow the best practices; **clear all resources** after you finished the work at that day.
- **Don’t worry**, when you come back tomorrow, we can instantly come the same situation with the power of **AWS CDK.**

https://www.youtube.com/watch?v=8_Xs8Ik0h1w

Creating our E-Commerce Serverless Microservices Project with CDK

Creating our E-Commerce Serverless Microservices Project with CDK

AWS Serverless Microservices with AWS Lambda Invocation Types

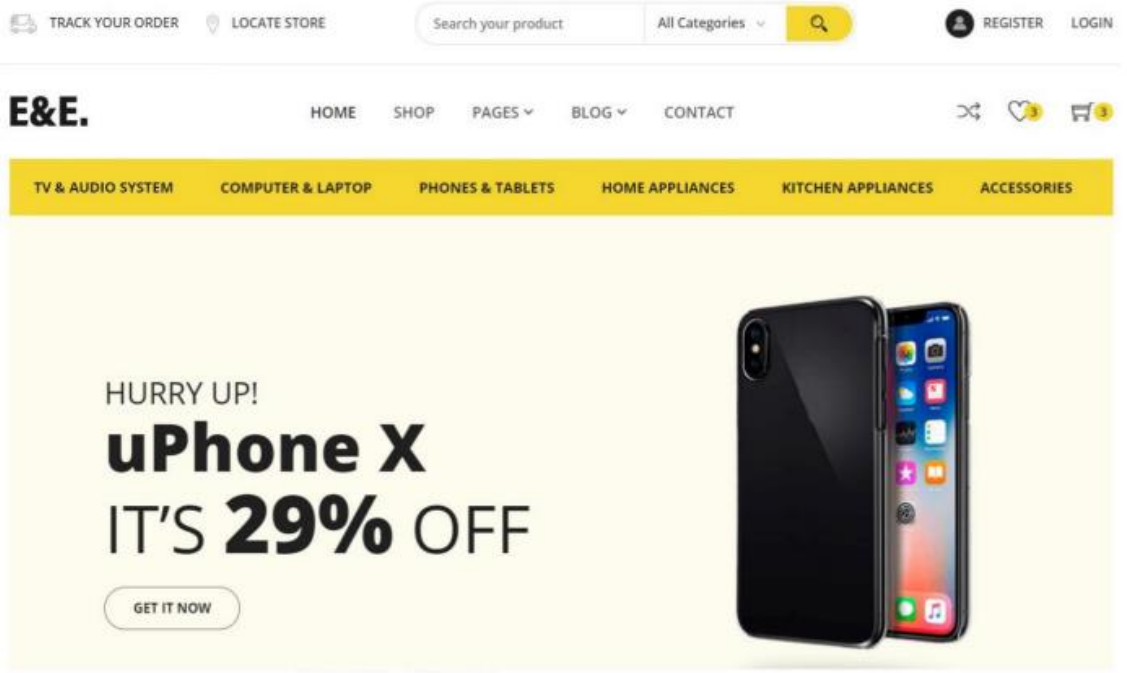


Understanding Our E-Commerce Domain

→ Understand E-Commerce Domain - Use Cases - Functional Requirement

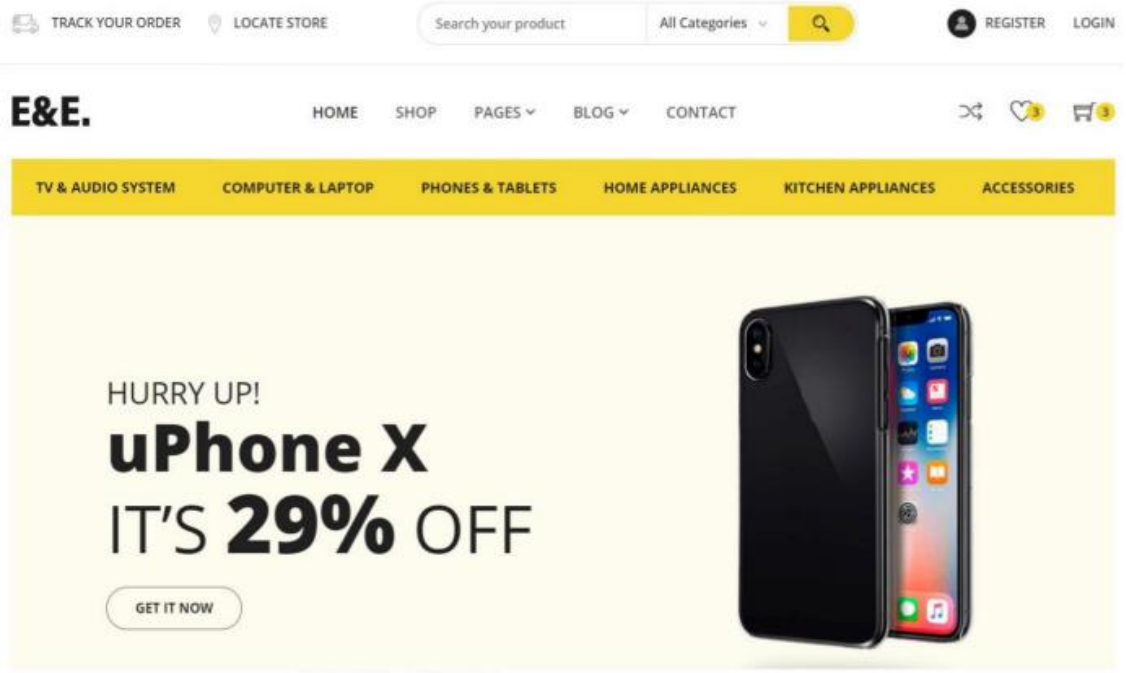
Understand E-Commerce Domain

- Use Cases
- Functional Requirement
- **Follow steps;**
- Requirements and Modelling
- Identify User Stories
- Identify the Nouns in the user stories
- Identify the Verbs in the user stories



E-Commerce Functional Requirements

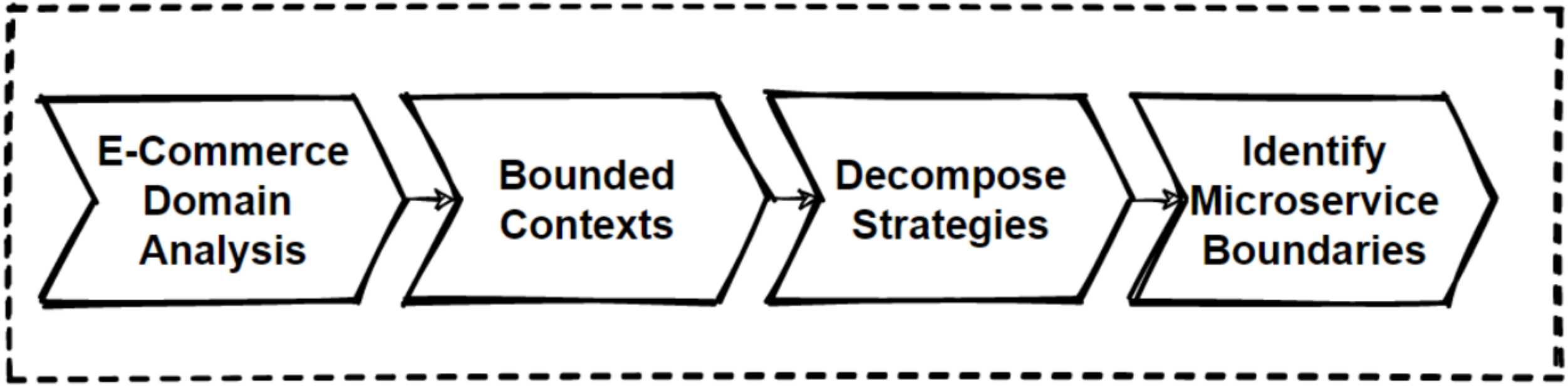
- List products
- Filter products as per categories
- Create - update - delete products - crud products
- Add item into the basket
- Checkout the basket and create a new order
- delete the basket
- List my orders and order items history
- Query my orders with orderDate



E-Commerce User Stories

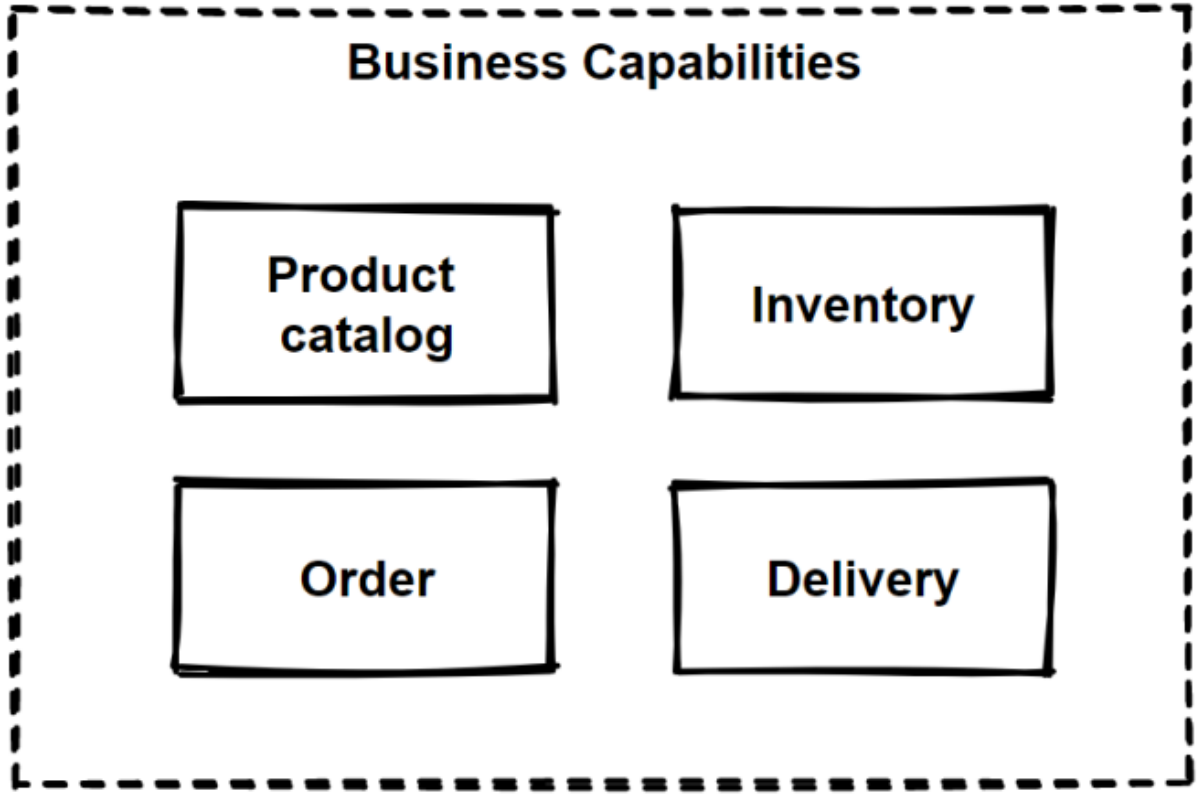
- As a user I want to list products
- As a user I want to filter products as per categories
- As a admin I want to Create - update - delete products - crud products
- As a user I want to add item into the basket so that I can check out quickly later on
- As a user I want to checkout the basket and create an order
- As a user I want to delete the basket
- As a user I want to list my orders and order items history
- As a user I want to Query my orders with orderDate

Decomposition Microservices Architecture Path



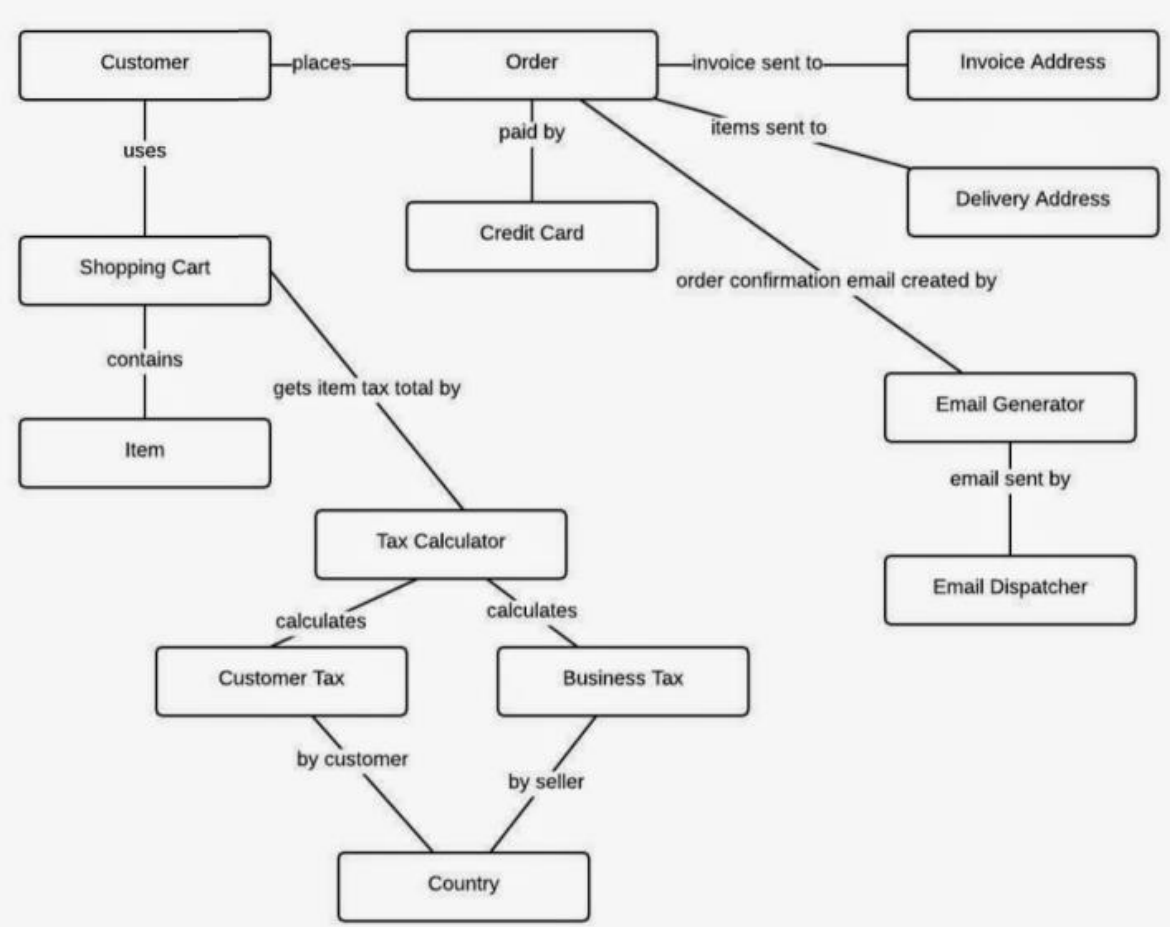
Microservices Decomposition Pattern – Decompose by Business Capability

- Split the application as a set of loosely coupled services
- Services must be **Cohesive**
- Services must be **Loosely Coupled**
- "**Decompose by Business Capability**" pattern offers that
  - Define services corresponding to business capabilities.
  - A business capability is a concept from business architecture modeling.
  - A business service should generate value.



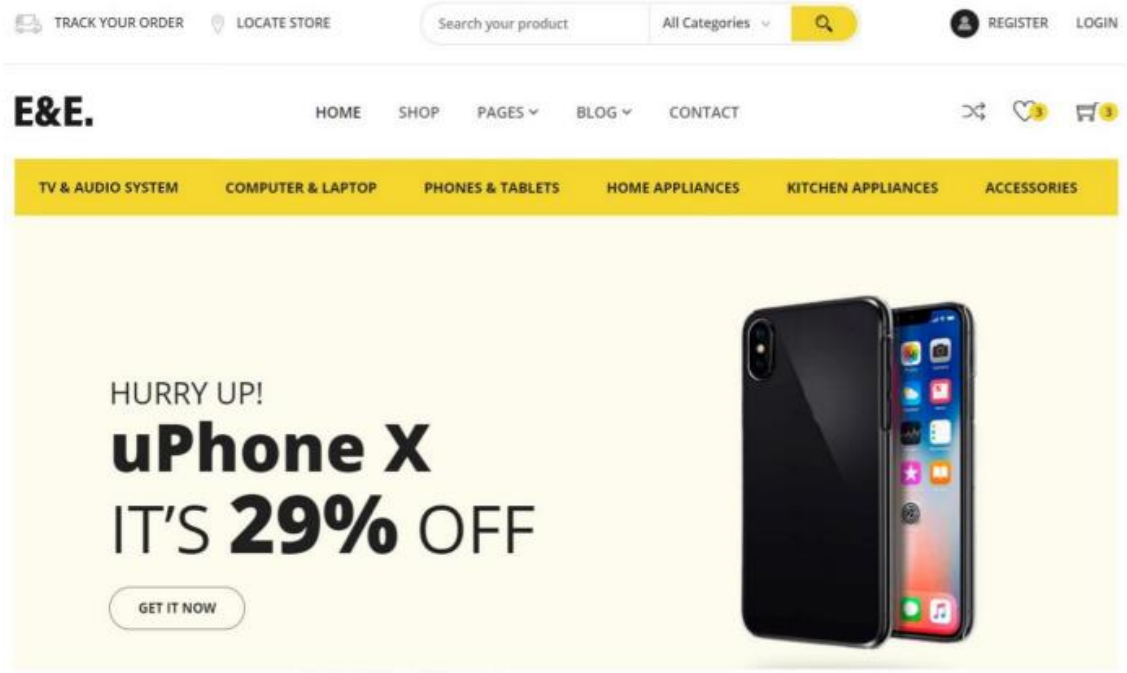
Analysis E-Commerce Domain - Nouns and Verbs

- List products
- Filter products as per categories
- Create - update - delete products - crud products
- Add item into the basket
- Checkout the basket and create a new order
- delete the basket
- List my orders and order items history
- Query my orders with orderDate



Analysis E-Commerce Domain - Nouns and Verbs

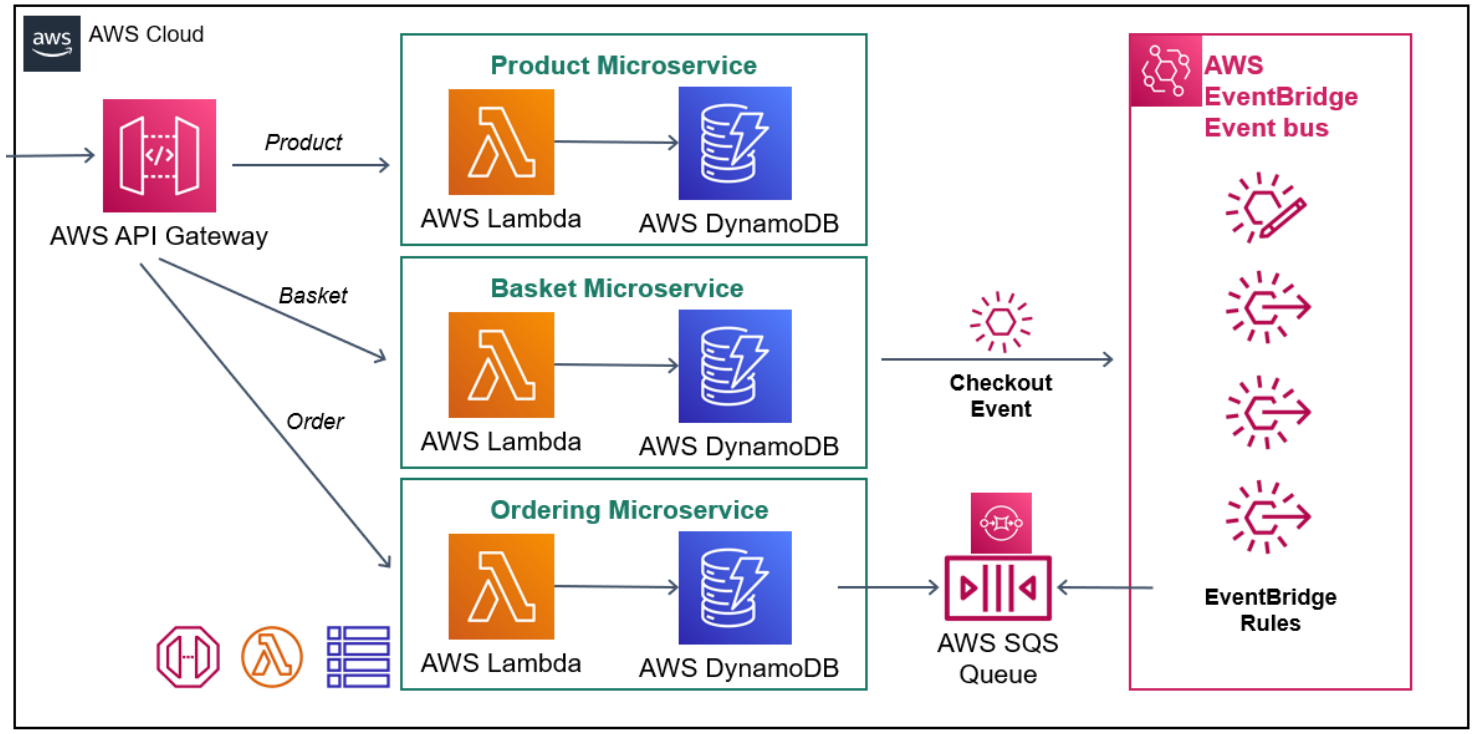
- As a user I want to **list products**
- As a user I want to **filter products** as per categories
- As a admin I want to **Create** - **update** - **delete products** - crud **products**
- As a user I want to **add item** into the **basket** so that I can check out quickly later on
- As a user I want to **checkout** the **basket** and **create** an **order**
- As a user I want to **delete** the **basket**
- As a user I want to **list** my **orders** and order items history
- As a user I want to **query** my **orders** with orderDate



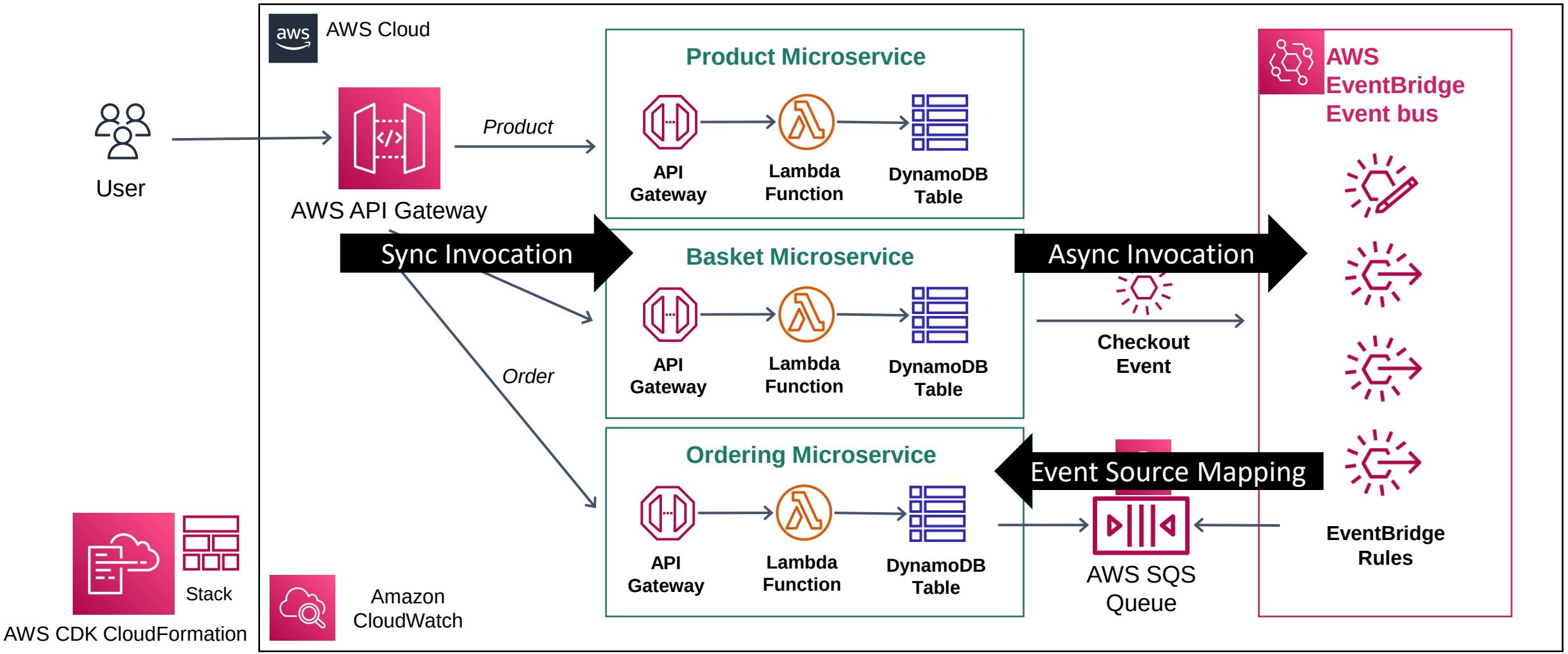
Green = Nouns = Microservices
Purple = Verbs = Integrations

Microservices Decomposition Pattern – Decompose by business capability

- Product
  - List products
  - Filter products as per categories
  - Create - update - delete products - crud products
- Basket
  - Add item into the basket
  - Checkout the basket and create a new order
  - Delete the basket
- Order
  - List my orders and order items history
  - Query my orders with orderDate



AWS Serverless Microservices with AWS Lambda Invocation Types

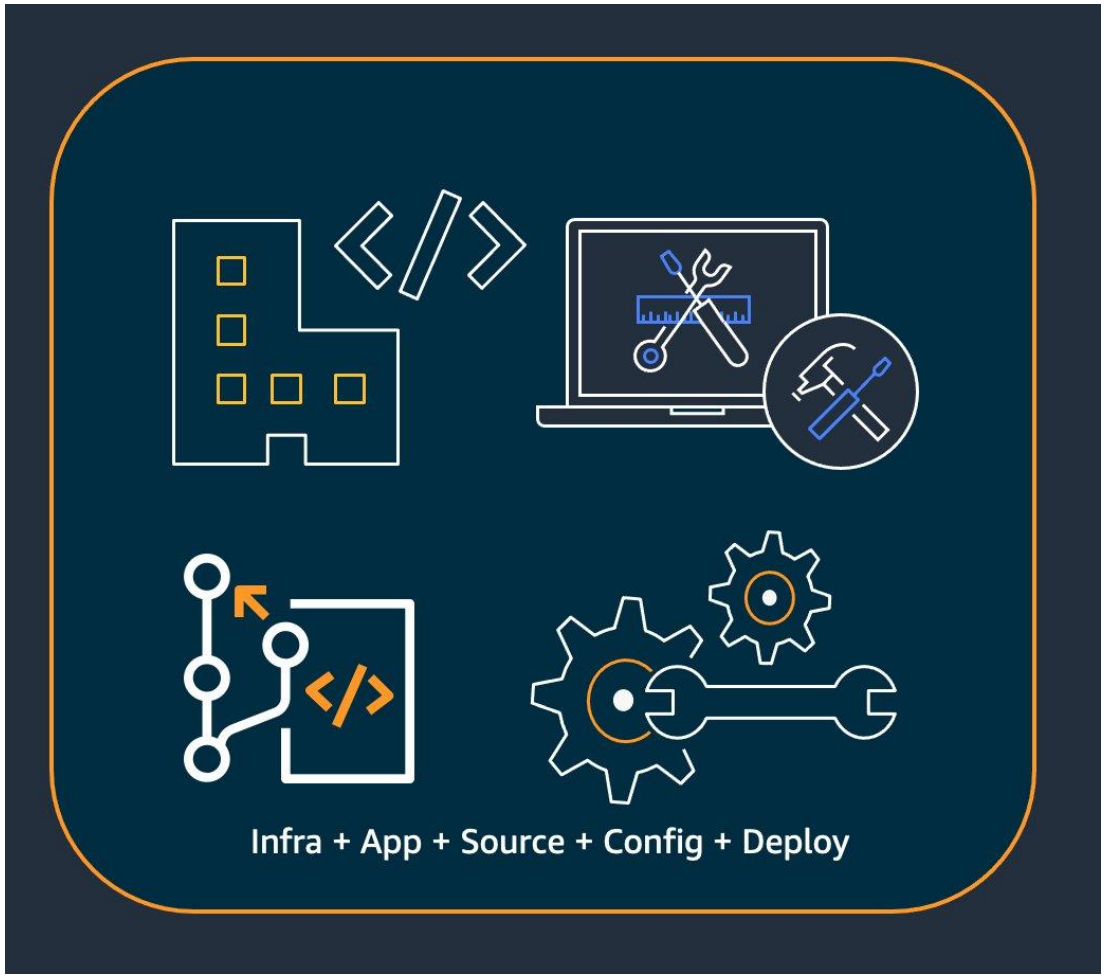


Code Structure of E-Commerce Serverless Microservices Project

→ Code Structure of E-Commerce Serverless Microservices Project in CDK

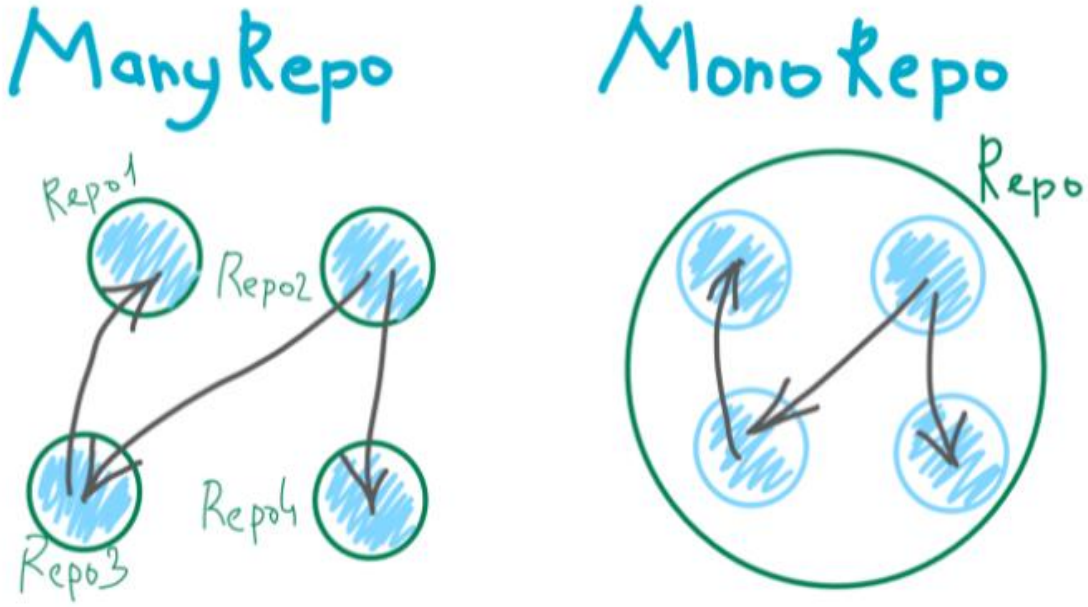
Code Structure of E-Commerce Serverless Microservices Project in CDK

- **Several ways** to organize your code.
- Think different way to **develop** and **deploy** your applications
- **Project structure** for Typescript applications
- Best practices for **developing** and **deploying** cloud infrastructure with the AWS CDK.
- Develop **E-Commerce Serverless** Microservices Project.
  - **IaC with CDK** - to develop this big architecture with CDK
  - **Lambda Microservice** business logic development with AWS SDK
- **Organize code** according to microservices -> product - basket and order microservices.
- **Best practice** for serverless applications is following **monorepo**.



What is Monorepo ?

- Software development strategy where many project codes are **stored** in the **same repository**
- **Single repository** that stores all of your code and assets for every project
- Companies like Google, Facebook, Microsoft, Uber, Airbnb, and Twitter
- **Single source of truth**, easy to **share code**, easy to refactor code
- Why We are using **Monorepo** ?
  - Microservices Part
  - Infrastructure Part
- Store **Microservices Code** and **Infrastructure Code** at the same repository.



https://medium.com/@jvr572/how-deploy-from-a-git-monorepo-1a9a55b23d44

2 Main Parts of Monorepo - Serverless E-commerce App

1

Serverless Infrastructure Development

IaC with AWS CDK using Typescript language

2

Microservices Lambda Function Development

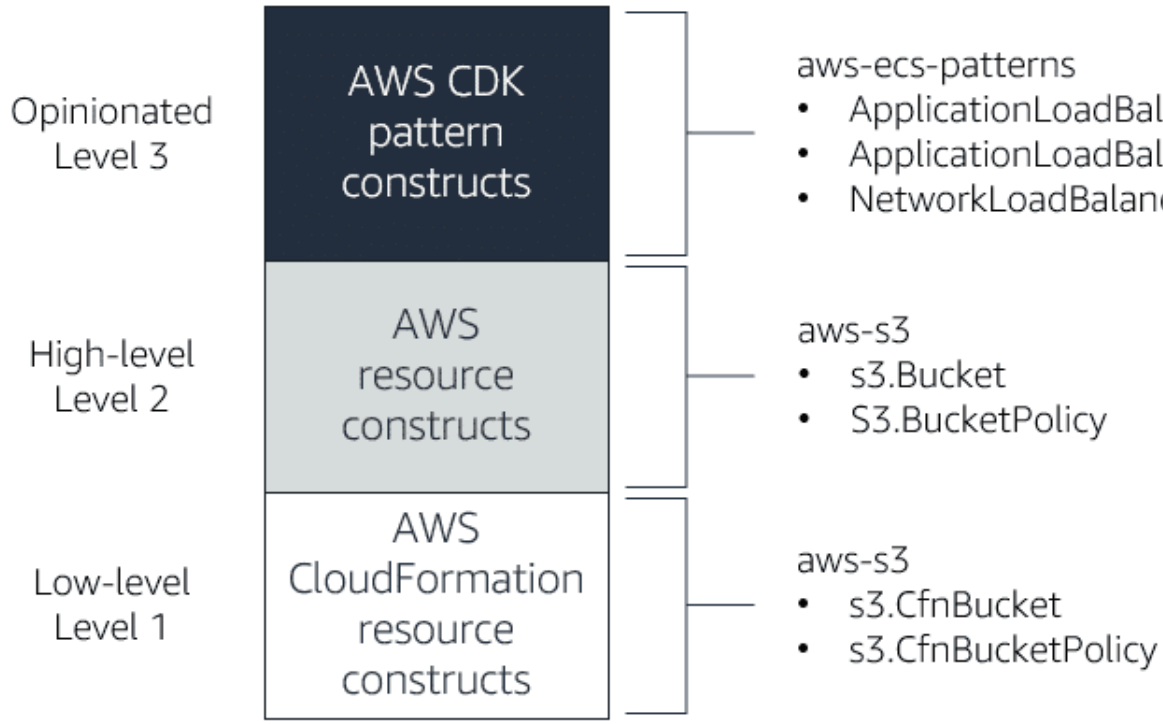
Nodejs Lambda Functions using AWS SDK for JavaScript v3

Project Folder Structure

- **Main Folders**
  bin, lib and src folders. bin/lib folders generate by aws cdk project template.
- **bin folder**
  Starting point of our application.
- **lib folder**
  Infrastructure codes. IaC Serverless Stacks with aws cdk.
- **src folder**
  Microservices development codes with nodejs.
- **DEMO** – Review Project Folder Structure into Visual Studio Code

AWS Solutions Constructs with Levels L1 - L2 - L3 Pattern Construct

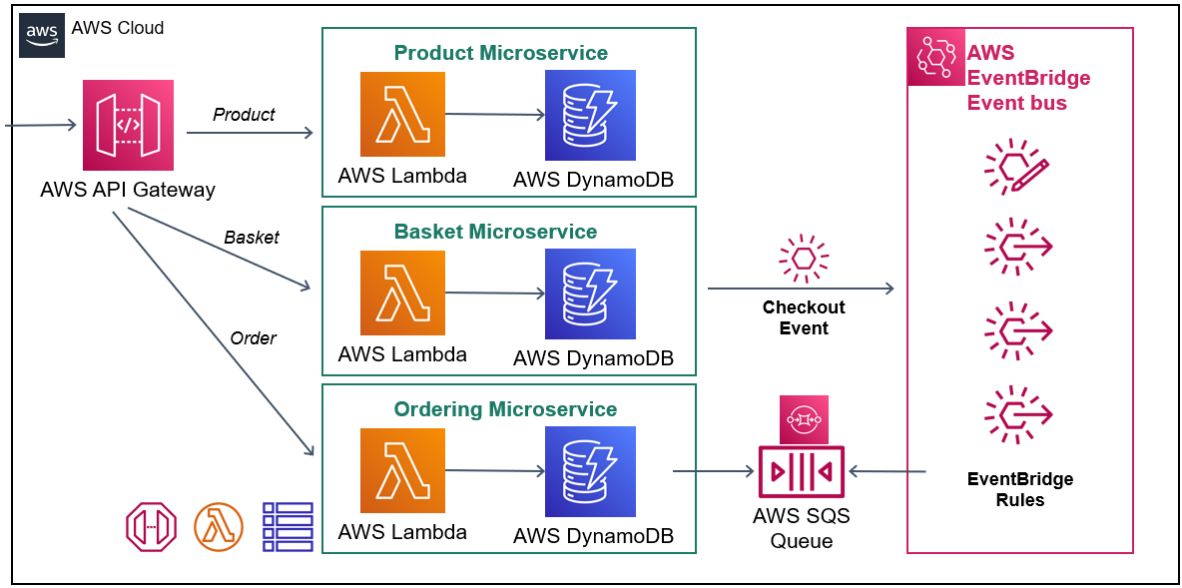
- **Level 1 (L1) construct**
  Direct representations of CloudFormation resources. Must provide all the required CloudFormation attributes. They are named CfnXyz.
- **Level 2 (L2) construct**
  Represents a particular cloud resource. S3 bucket. You don't have to configure every configuration attribute. higher-level, intent-based API. s3.Bucket class represents an Amazon S3, such as bucket.addLifeCycleRule()
- **Level 3 (L3) construct = Pattern Construct**
  Represents a bunch of cloud resources that work together to accomplish a particular task. You can create an ApplicationLoadBalancedFarageteService. Higher-level constructs, which we call patterns.



https://docs.aws.amazon.com/cdk/v2/guide/home.html

AWS CDK (IaC) vs AWS SDK (Microservices)

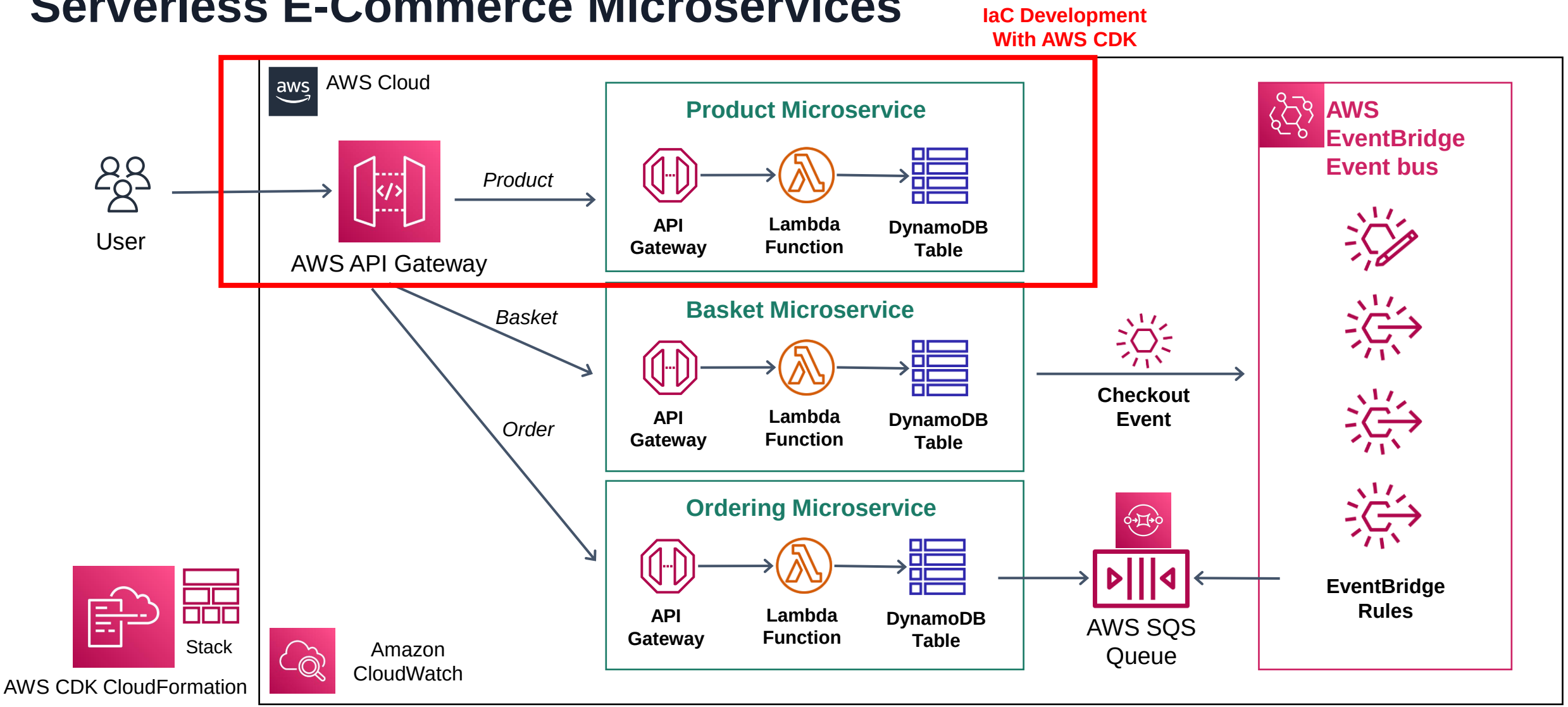
- **AWS SDK - Software Development Kit**
  Simplifies use of AWS Services by providing a set of libraries that are consistent and familiar for developers.
- Tools for developing and managing applications on AWS
- Use **AWS-SDK** in our microservices codes when interacting with AWS DynamoDB, EventBridge and SQS.
- **AWS CDK - IaC**
  Infrastructure as code (IaC) service that allows you to easily model, provision, and manage AWS resources.
- **When we use AWS CDK and AWS SDK ?**
  - IaC with CDK - to develop this big architecture with CDK
  - Lambda Microservice business logic development with AWS SDK



Creating Product Microservices Serverless CRUD REST API

→ Creating Product Microservices Serverless CRUD REST API Infrastructure with AWS CDK (Api Gateway - Lambda - DynamoDB)

Serverless E-Commerce Microservices



Prerequisites 6 - Docker Desktop

- For **Bundling Lambda Function** with Libraries.
- Required from aws cdk when you develop lambda fuctions which include external libraries.
- Installing or updating the latest version of the Docker
- Successfuly installed Docker Desktop.
- Check all versions;
- aws --version
- node --version
- npm --version
- cdk --version
- docker --version

https://www.docker.com/products/docker-desktop

Prerequisites 7 - Postman

- For **Test** and **Manage ApiGateway Apis**.
- Installing or updating the latest version of the Postman
- Successfuly installed Postman.
- Create postman collection for our all microservices apis.
- Manage urls with environment variables according to environment deployments.
- test our Product Microservices-Serverless CRUD REST API with Postman.

https://www.postman.com/

Clean up Resources

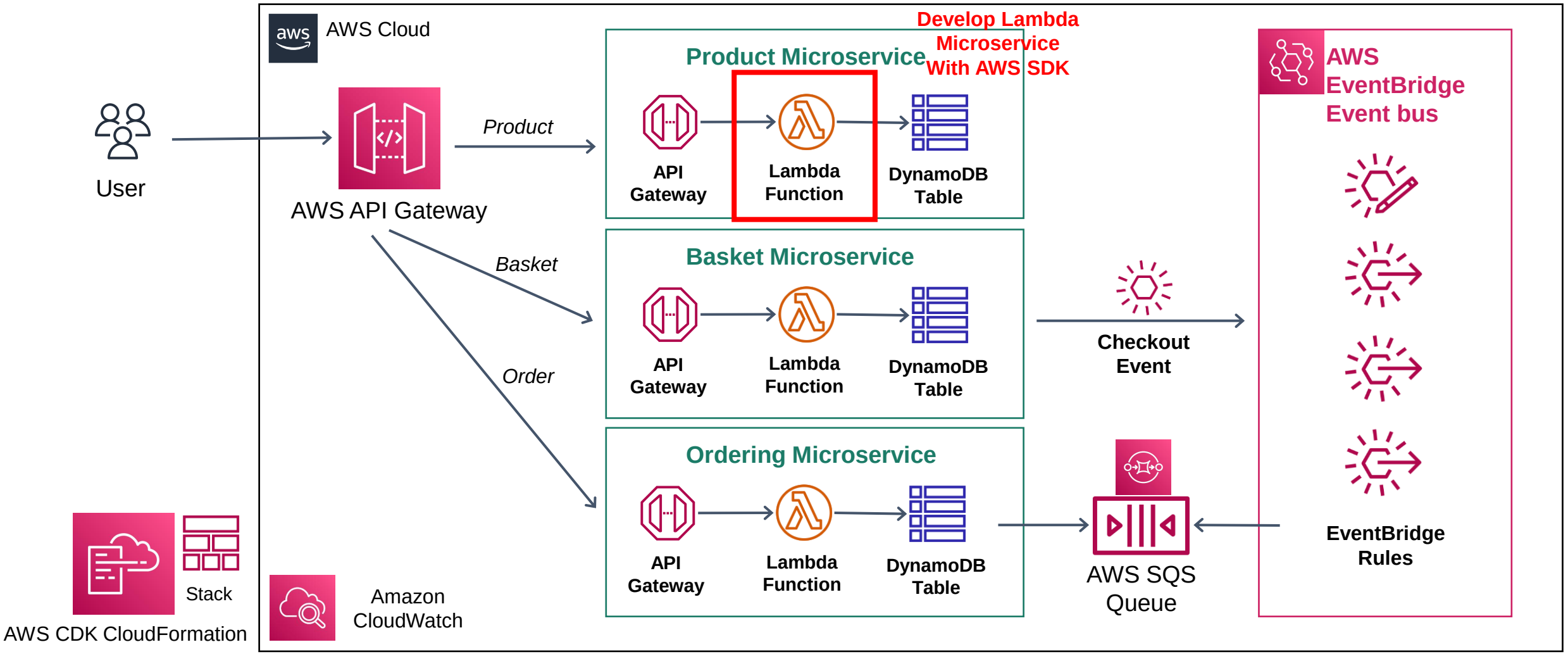
- **Amazon CTO Werner Vogels**
  Turn off the lights before leaving the room
- If you leave from computer and shut the light off, that means you also **shut down the AWS Resources**
- Think that we are going to prepare **demo session** with **your customer**
- Follow the best practices; **clear all resources** after you finished the work at that day.
- **Don’t worry**, when you come back tomorrow, we can instantly come the same situation with the power of **AWS CDK.**

https://www.youtube.com/watch?v=8_Xs8Ik0h1w

Developing Product Lambda Microservices CRUD functions

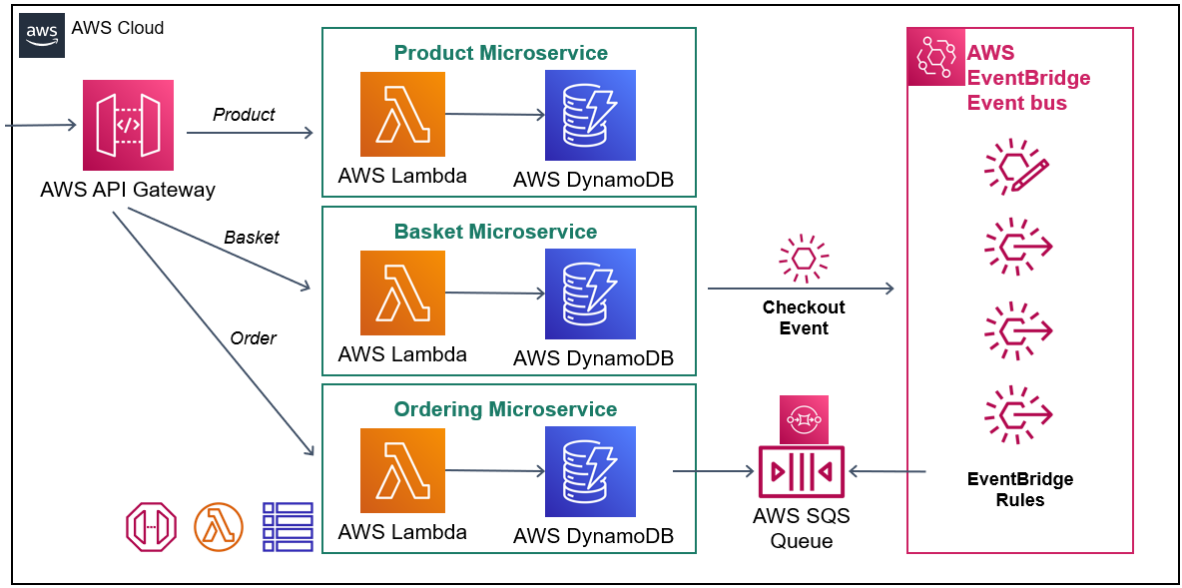
→ Developing Product Lambda Microservices CRUD functions with AWS SDK

Serverless E-Commerce Microservices



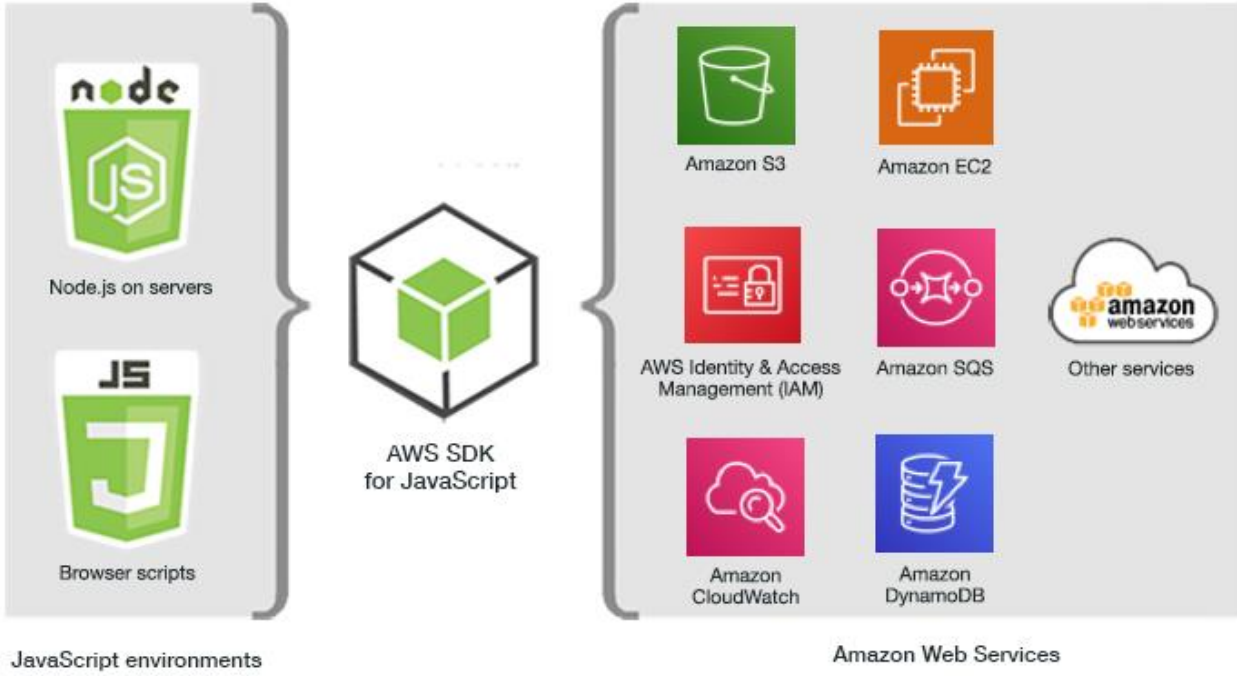
AWS CDK (IaC) vs AWS SDK (Microservices)

- **AWS SDK - Software Development Kit**
  Simplifies use of AWS Services by providing a set of libraries that are consistent and familiar for developers.
- Tools for developing and managing applications on AWS
- Use **AWS-SDK** in our microservices codes when interacting with aws dynamodb, eventbridge and sqs.
- **AWS CDK - IaC**
  Infrastructure as code (IaC) service that allows you to easily model, provision, and manage AWS resources.
- **When we use AWS CDK and AWS SDK ?**
  - IaC with CDK - to develop this big architecture with CDK
  - Lambda Microservice business logic development with AWS SDK



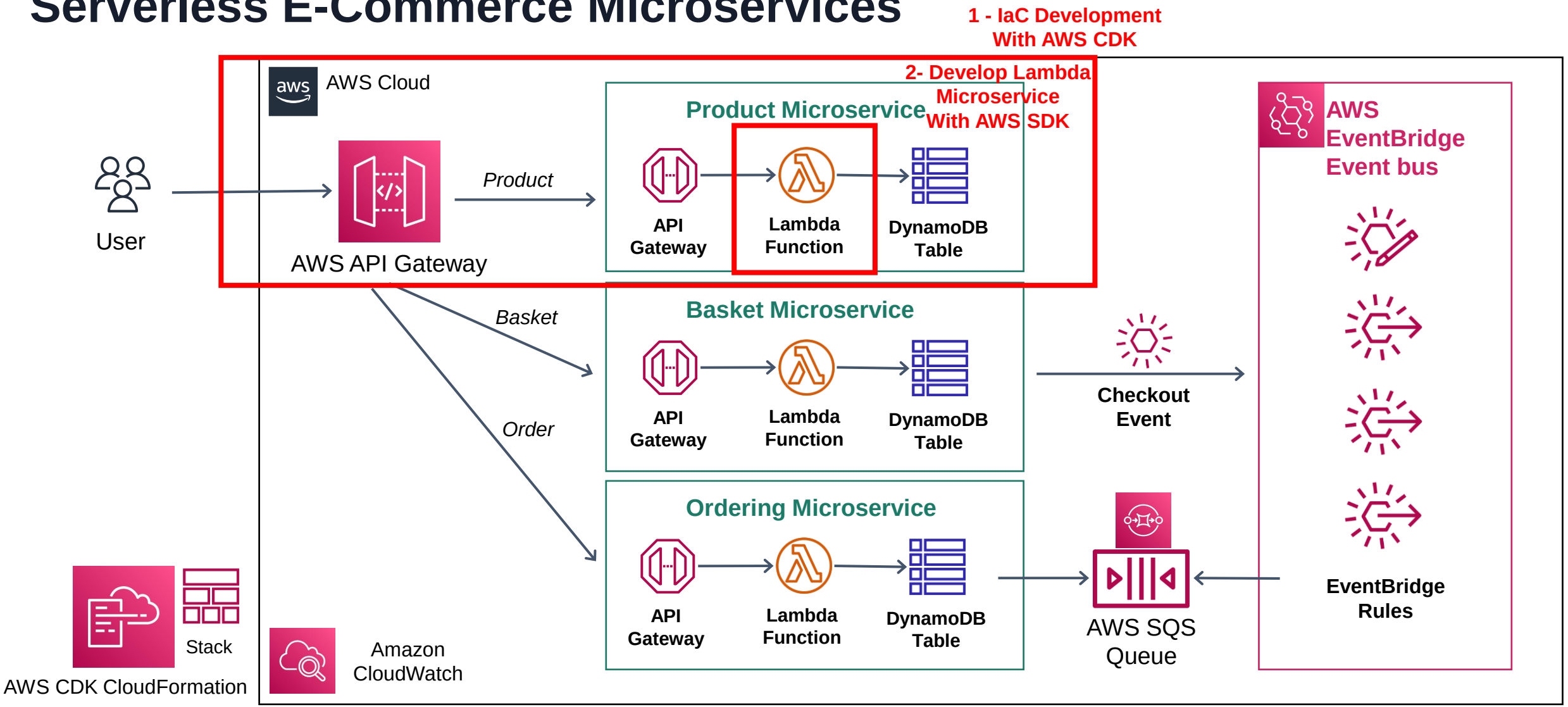
AWS SDK for JavaScript Version 3 and Lambda Interactions

- **Product Lambda Microservice** Function is chosen the runtime as a **NodeJS** function.
- Use **AWS SDK** for **JavaScript Version 3**.
- First-class **TypeScript** support and a new middleware stack
- We have 2 main resources.
- [AWS SDK for JavaScript - Developer Guide for SDK Version 3]
- [AWS SDK for JavaScript v3 API Reference Guide]



https://aws.amazon.com/blogs/developer/modular-aws-sdk-for-javascript-is-now-generally-available/

Serverless E-Commerce Microservices



Clean up Resources

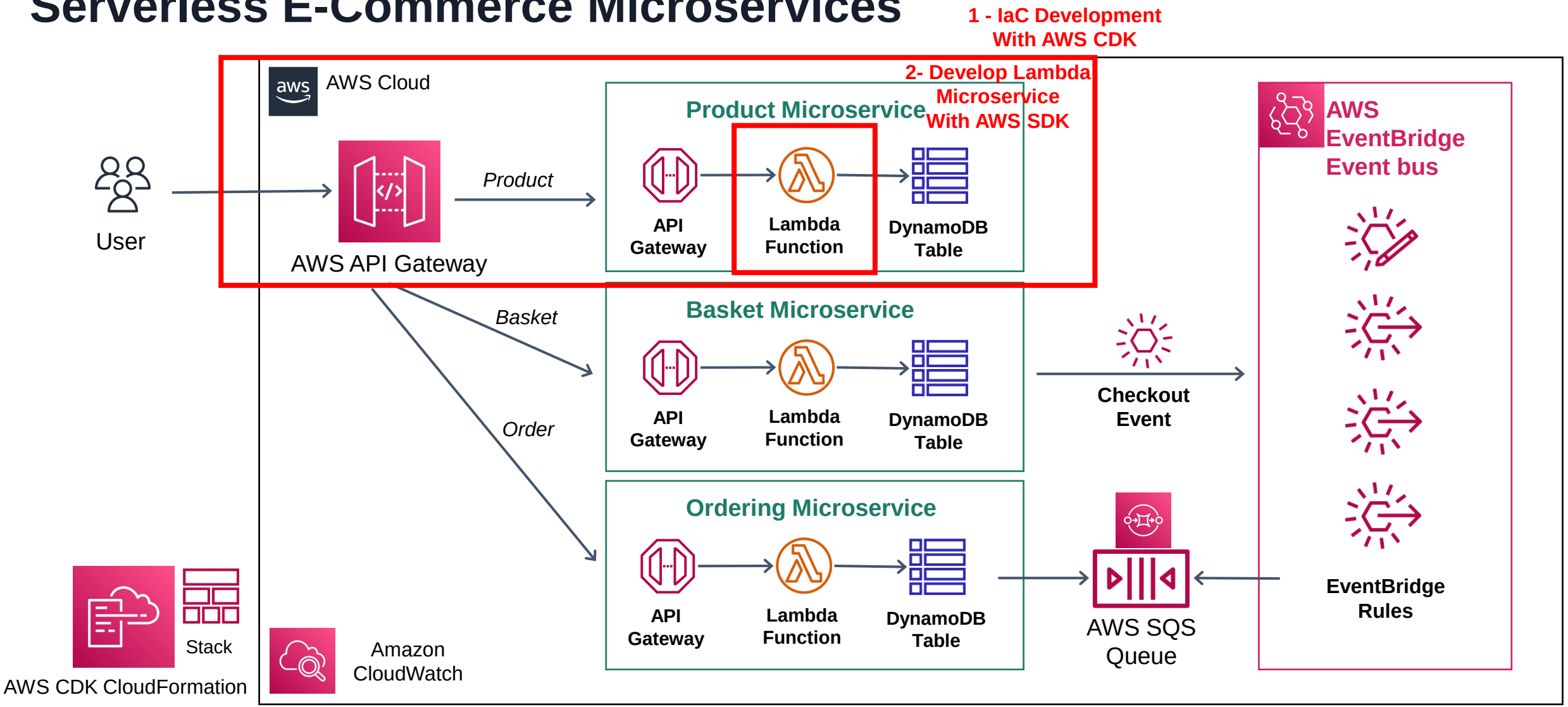
- **Amazon CTO Werner Vogels**
  Turn off the lights before leaving the room
- If you leave from computer and shut the light off, that means you also **shut down the AWS Resources**
- Think that we are going to prepare **demo session** with **your customer**
- Follow the best practices; **clear all resources** after you finished the work at that day.
- **Don’t worry**, when you come back tomorrow, we can instantly come the same situation with the power of **AWS CDK.**

https://www.youtube.com/watch?v=8_Xs8Ik0h1w

Organize our Infrastructure Code with Creating our Solution Constructs on AWS CDK

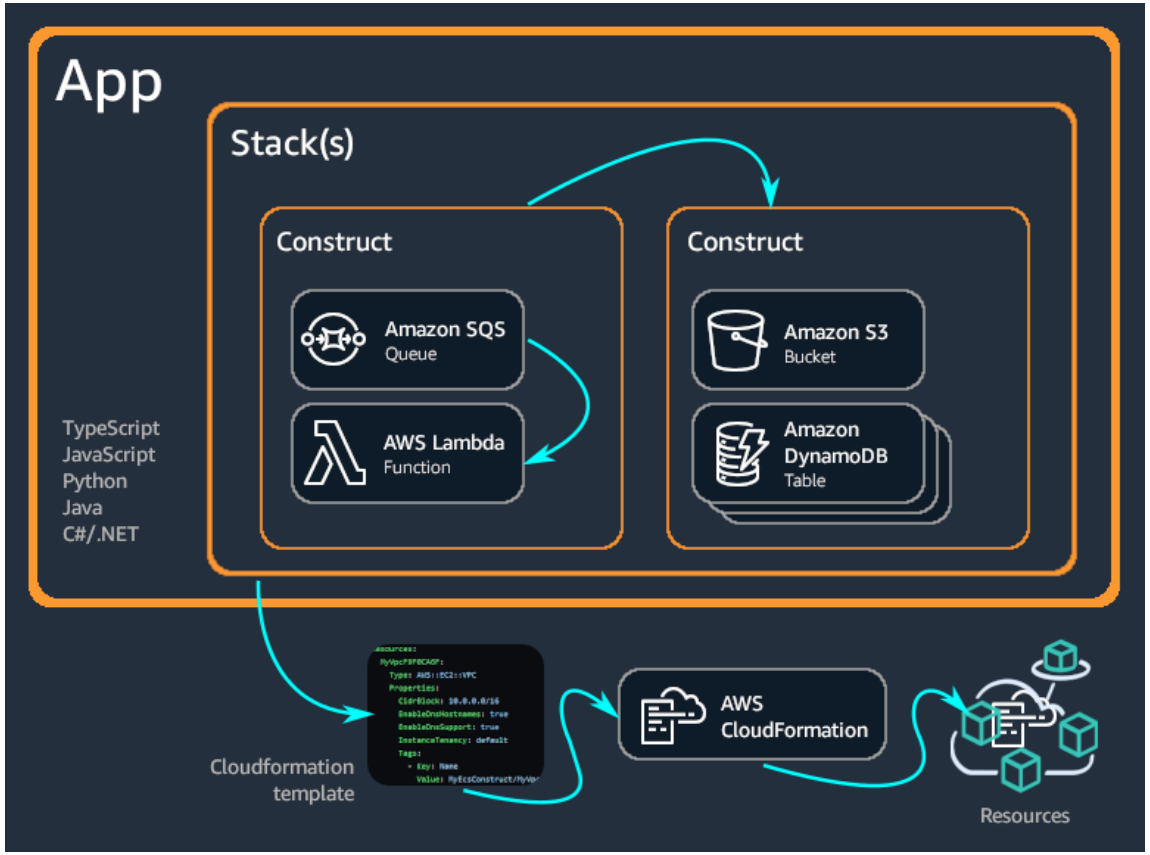
→ Organize our Infrastructure Code with Creating our Solution Constructs on AWS CDK

Serverless E-Commerce Microservices



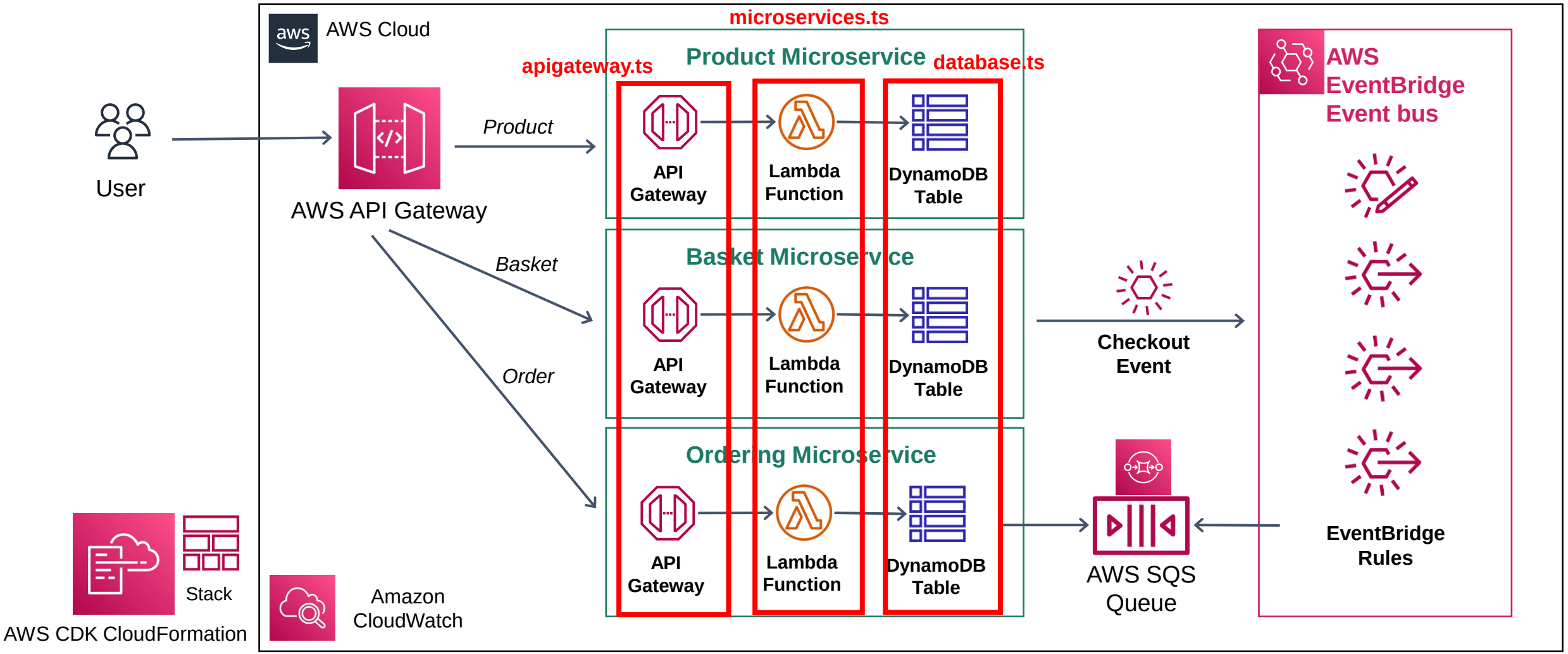
AWS CDK Core Concepts - Apps - Stacks - Constructs - Environments

- **Apps**
  Include everything needed to deploy your app to a cloud environment.
- **Stack**
  The unit of deployment in the AWS CDK is called a stack.
- **Constructs**
  The basic building blocks of AWS CDK apps. A construct represents a “cloud component”.
- **Environments**
  Each Stack instance in your AWS CDK app is explicitly or implicitly associated with an environment.



https://docs.aws.amazon.com/cdk/v2/guide/home.html

Serverless E-Commerce Microservices



Clean up Resources

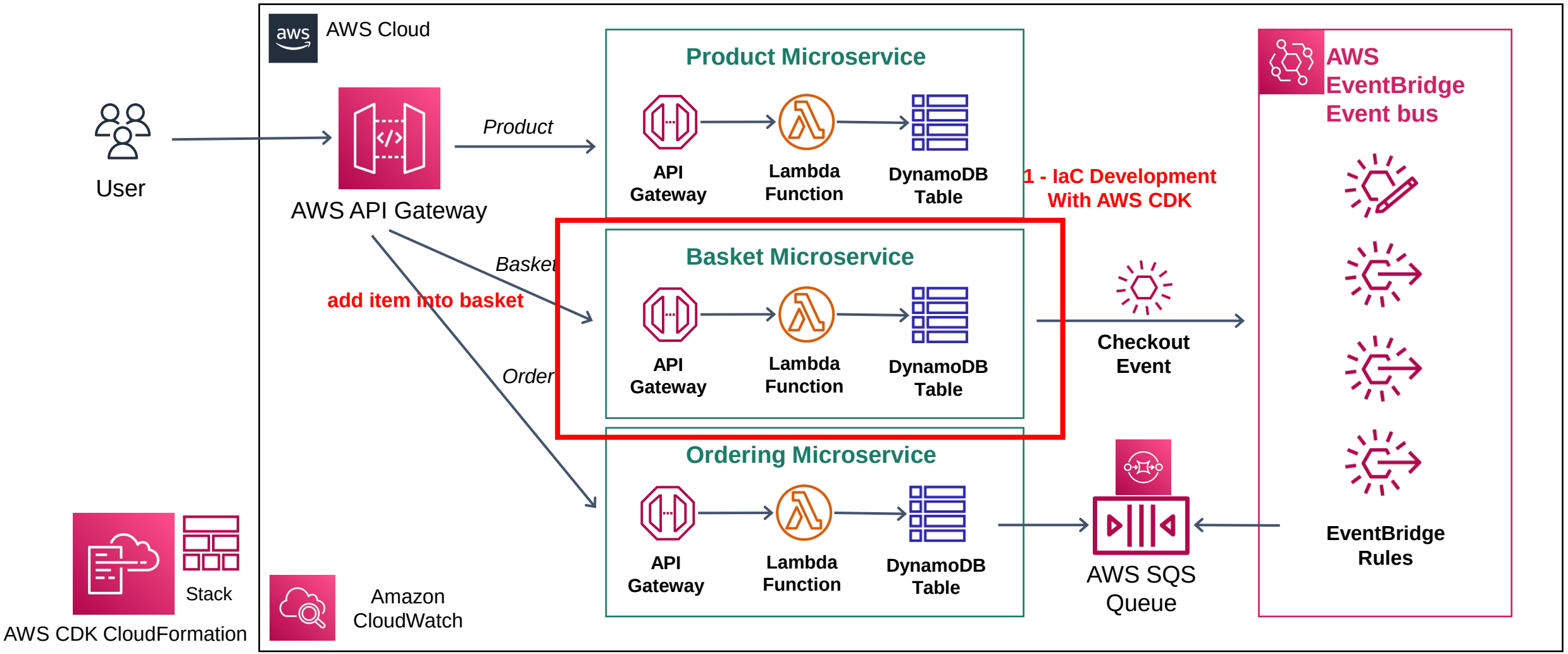
- **Amazon CTO Werner Vogels**
  Turn off the lights before leaving the room
- If you leave from computer and shut the light off, that means you also **shut down the AWS Resources**
- Think that we are going to prepare **demo session** with **your customer**
- Follow the best practices; **clear all resources** after you finished the work at that day.
- **Don’t worry**, when you come back tomorrow, we can instantly come the same situation with the power of **AWS CDK.**

https://www.youtube.com/watch?v=8_Xs8Ik0h1w

Creating Basket Microservices Infrastructure with AWS CDK

→ Creating Basket Microservices Infrastructure with AWS CDK - Add to Basket Sync Flow

Serverless E-Commerce Microservices



Clean up Resources

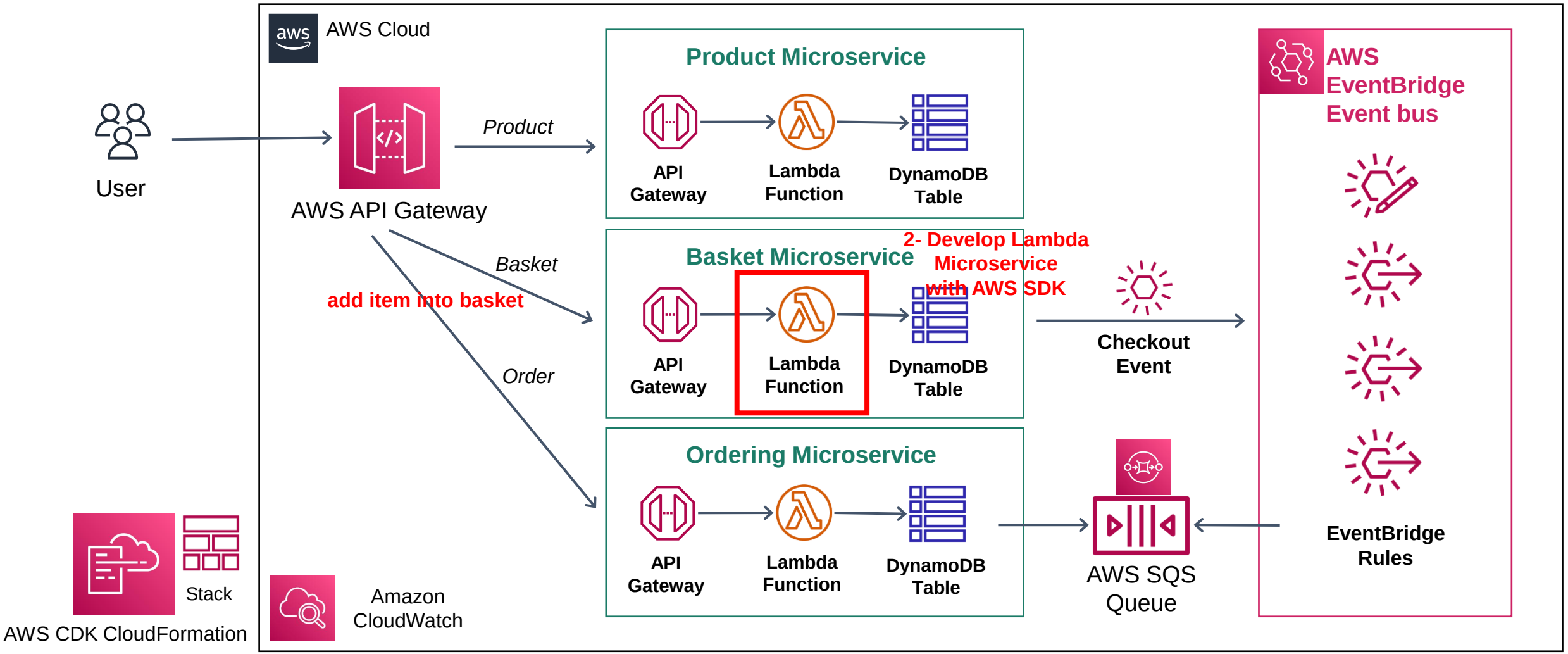
- **Amazon CTO Werner Vogels**
  Turn off the lights before leaving the room
- If you leave from computer and shut the light off, that means you also **shut down the AWS Resources**
- Think that we are going to prepare **demo session** with **your customer**
- Follow the best practices; **clear all resources** after you finished the work at that day.
- **Don’t worry**, when you come back tomorrow, we can instantly come the same situation with the power of **AWS CDK.**

https://www.youtube.com/watch?v=8_Xs8Ik0h1w

Developing Basket Lambda Microservices functions with AWS SDK

- Build a add/remove basket API for DynamoDB w/ AWS Lambda & API Gateway using Node.js AWS-SDK V3 & Serverless.
- Develop checkout basket async use cases.

Serverless E-Commerce Microservices



Clean up Resources

- **Amazon CTO Werner Vogels**
  Turn off the lights before leaving the room
- If you leave from computer and shut the light off, that means you also **shut down the AWS Resources**
- Think that we are going to prepare **demo session** with **your customer**
- Follow the best practices; **clear all resources** after you finished the work at that day.
- **Don’t worry**, when you come back tomorrow, we can instantly come the same situation with the power of **AWS CDK.**

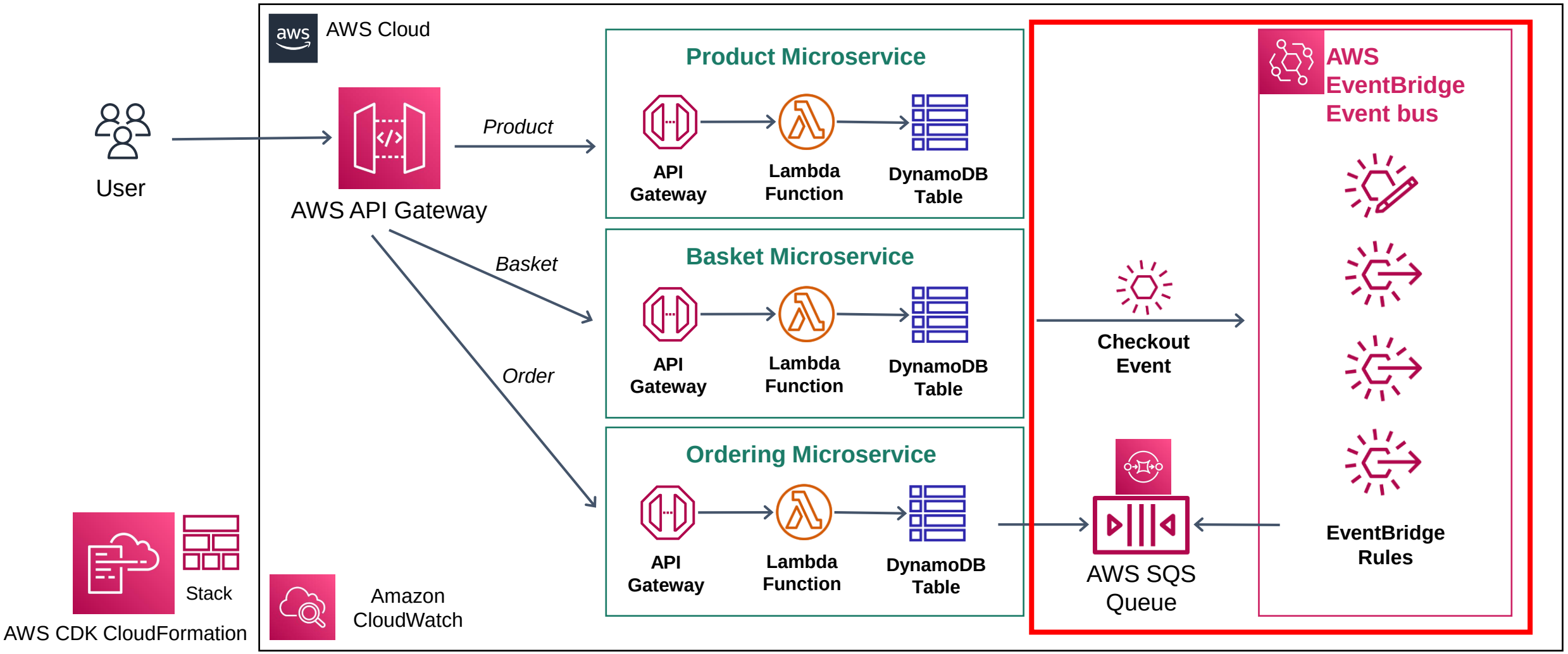
https://www.youtube.com/watch?v=8_Xs8Ik0h1w

AWS EventBridge for Decouple Microservices

- AWS EventBridge for Decouple Microservices with Event-Driven Architecture.

  Use EventBridge as a Event Bus for Decoupling basket and ordering microservices.

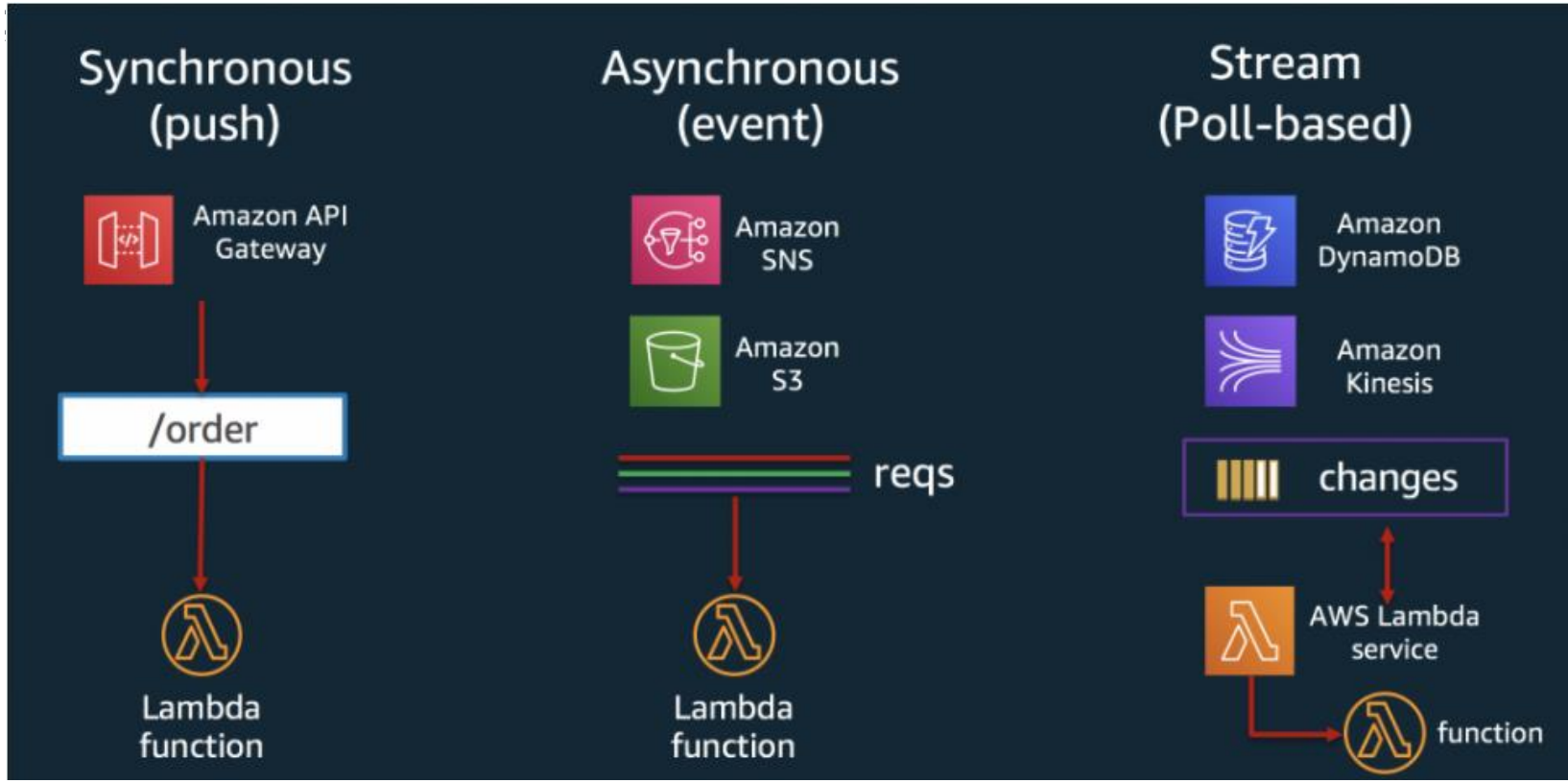
Serverless E-Commerce Microservices



AWS Lambda Invocation Types

- Triggered lambda functions with different AWS Lambda Invocation Types
- AWS Lambda has 3 Invocation Types;
- **Lambda Synchronous invocation**
- **Lambda Asynchronous invocation**
- **Lambda Event Source Mapping**

with polling invocation



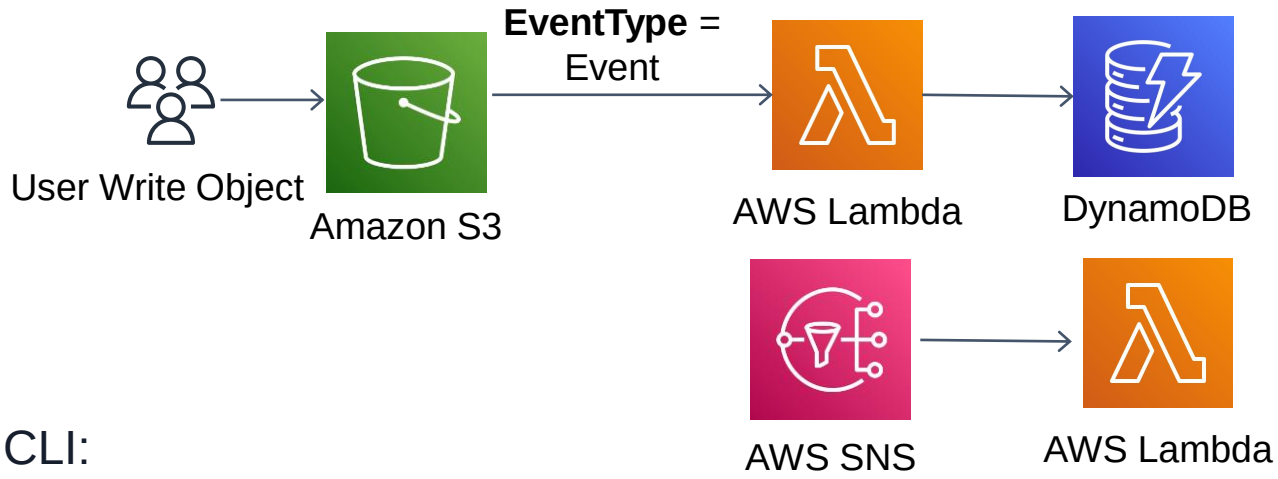
https://aws.amazon.com/blogs/architecture/understanding-the-different-ways-to-invoke-lambda-functions/

AWS Lambda Asynchronous Invocation

- Lambda **sends** the **event** to a **internal queue** and returns a **success response** without any additional information
- Separate process **reads events** from the **queue** and **runs** our lambda function
- **S3 / SNS + Lambda + DynamoDB**
- **Invocation-type** flag should be **“Event”**
- AWS Lambda sets a **retry policy**

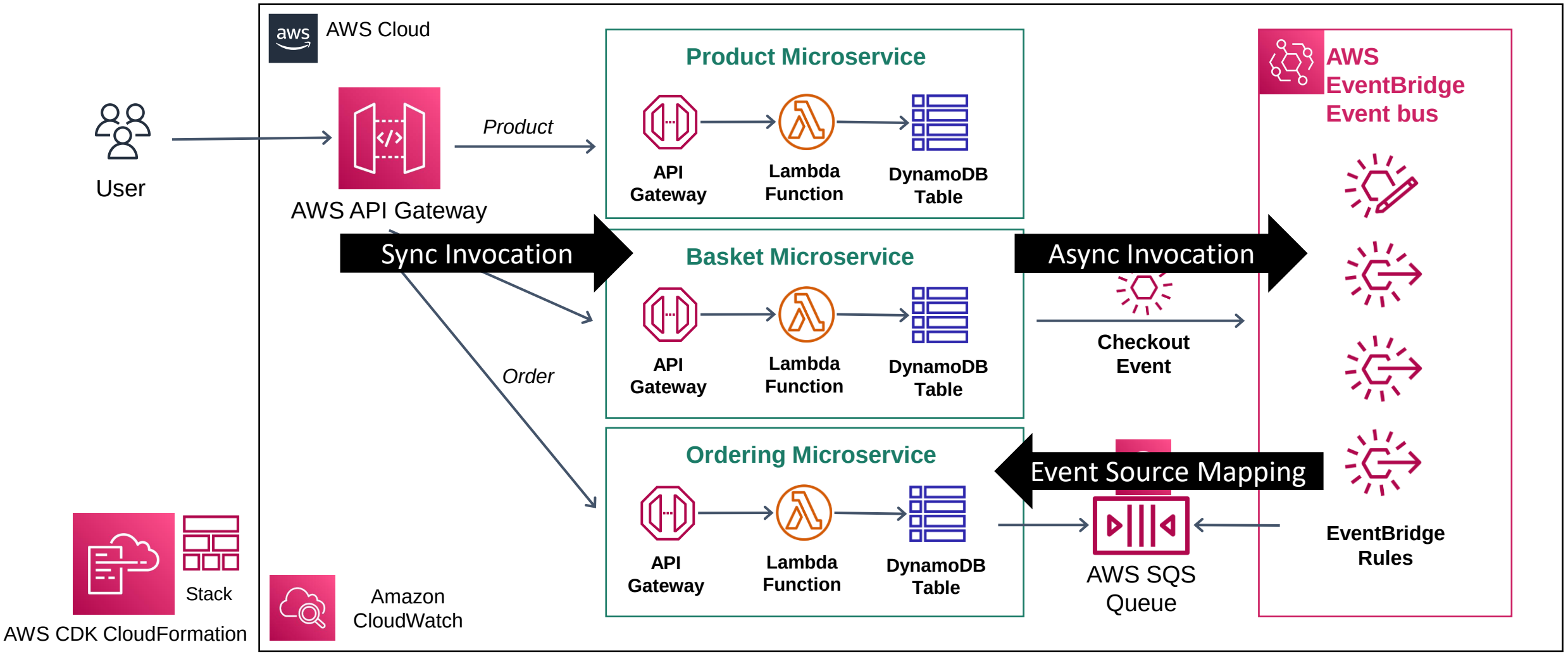
  Retry Count = 2

  Attach a Dead-Letter Queue (DLQ)



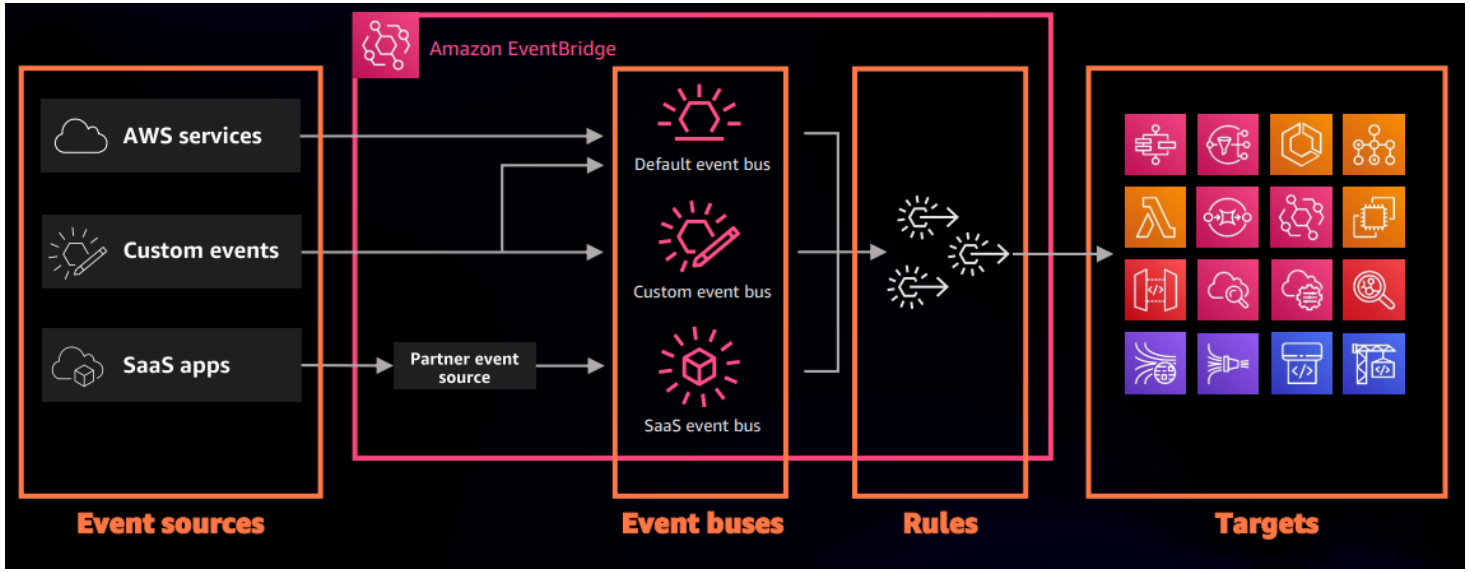
- Example of asynchronous invocation using the AWS CLI:
  aws lambda invoke —function-name MyLambdaFunction **—invocation-type Event** —payload '{ "key": "value" }'
- Triggered **AWS services** of **asynchronous invocation**; S3, EventBridge, SNS, SES, CloudFormation, CloudWatch Logs, CloudWatch Events, CodeCommit

AWS Serverless Microservices with AWS Lambda Invocation Types



What is Amazon EventBridge ?

- **Serverless event bus** service for AWS services
- Build **event-driven applications** at scale using events generated from your apps
- Use to **connect** your applications with data from a variety of sources, **integrated SaaS applications**
- AWS services to **targets** such as **AWS Lambda functions**
- Formerly called Amazon CloudWatch Events



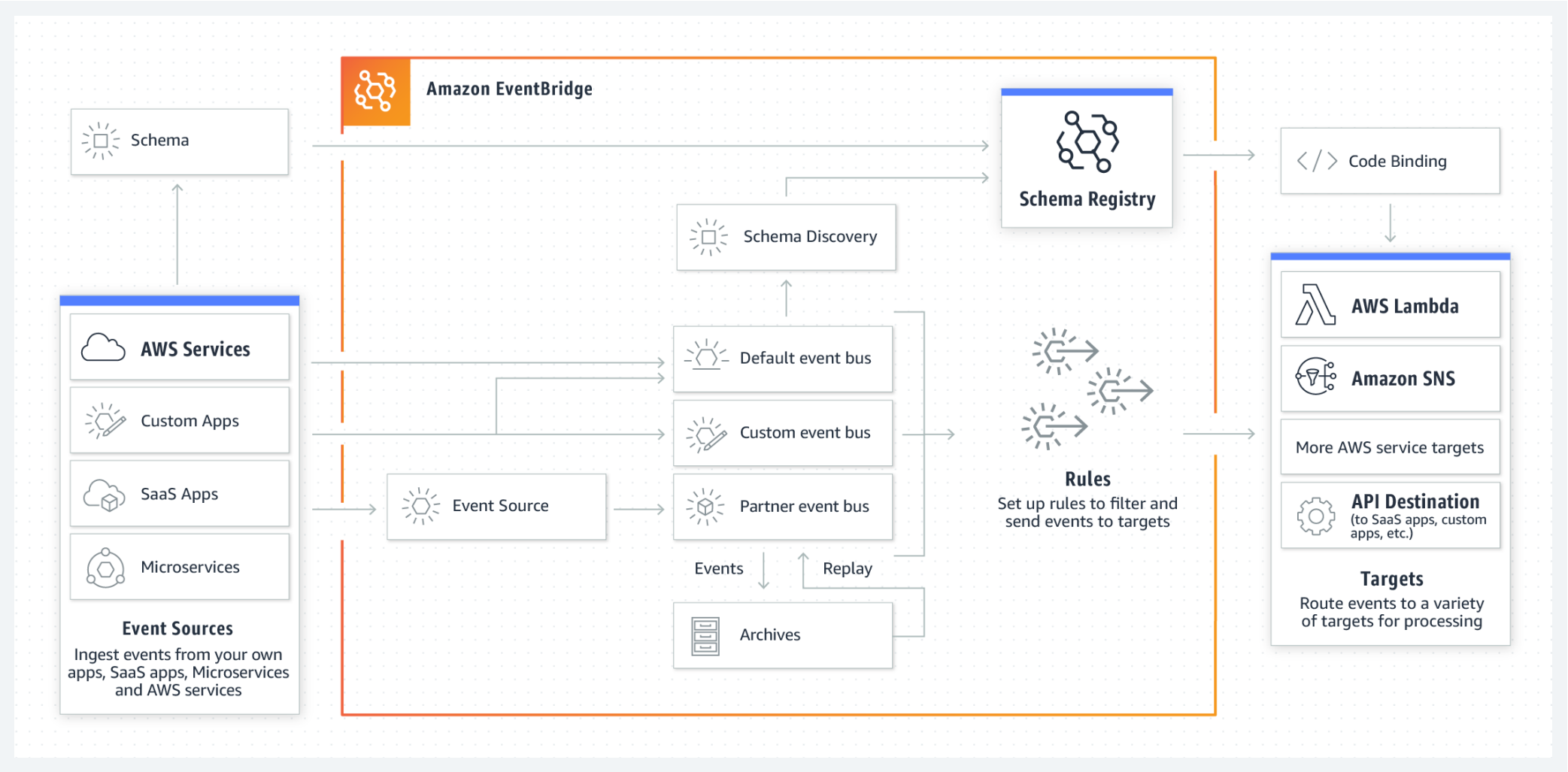
https://da-public-assets.s3.amazonaws.com/serverlessland/pdf/2021+-+Serverlesspresso+exhibit+-+PDF.pdf

Benefits of Amazon EventBridge

Amazon EventBridge

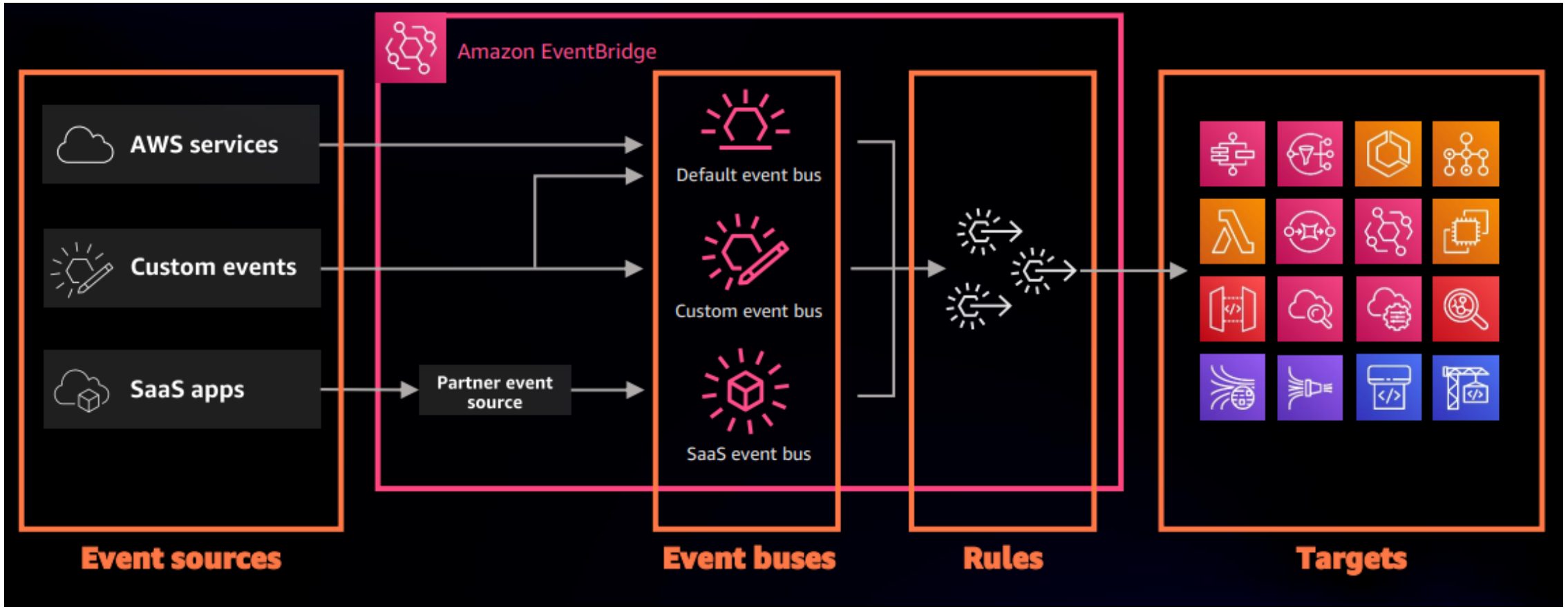
- **Build event-driven architectures**
  With EventBridge, your event targets don’t need to be aware of event sources because you can filter and publish directly to EventBridge. Improves developer agility as well as application resiliency with loosely coupled event-driven architectures.
- **Connect SaaS apps**
  EventBridge ingests data from supported SaaS applications and routes it to AWS services and SaaS targets. SaaS apps to trigger workflows for customer support, business operations.
- **Write less custom code**
  You can ingest, filter, transform and deliver events without writing custom code. The EventBridge schema registry stores a collection of easy-to-find event schemas.
- **Reduce operational overhead**
  There are no servers to provision, patch, and manage. automatically scales based on the number of events ingested. Built-in distributed availability and fault-tolerance. Native event archive and replay capability.

How Amazon EventBridge works ?



https://aws.amazon.com/eventbridge/

How Amazon EventBridge works ?



https://da-public-assets.s3.amazonaws.com/serverlessland/pdf/2021+-+Serverlesspresso+exhibit+-+PDF.pdf

EventBridge Concepts - Events - Event Buses - Rules - Targets

- **Amazon EventBridge Events**
  An event indicates a change in an environment such as an AWS environment or a SaaS partner service. Events are represented as JSON objects and they all have a similar structure, and the same top-level fields.
- **Amazon EventBridge Rules**
  A rule matches incoming events and sends them to targets for processing. A single rule can send an event to multiple targets, which then run in parallel. An event pattern defines the event structure and the fields that a rule matches.
- **Amazon EventBridge Targets**
  A target is a resource or endpoint that EventBridge sends an event to when the event matches the event pattern defined for a rule. The rule processes the event data and sends the relevant information to the target.
- **Amazon EventBridge Event Buses**
  An event bus is a pipeline that receives events. Rules associated with the event bus evaluate events as they arrive. A resource-based policy specifies which events to allow, and which entities have permission to create or modify rules or targets for an event.

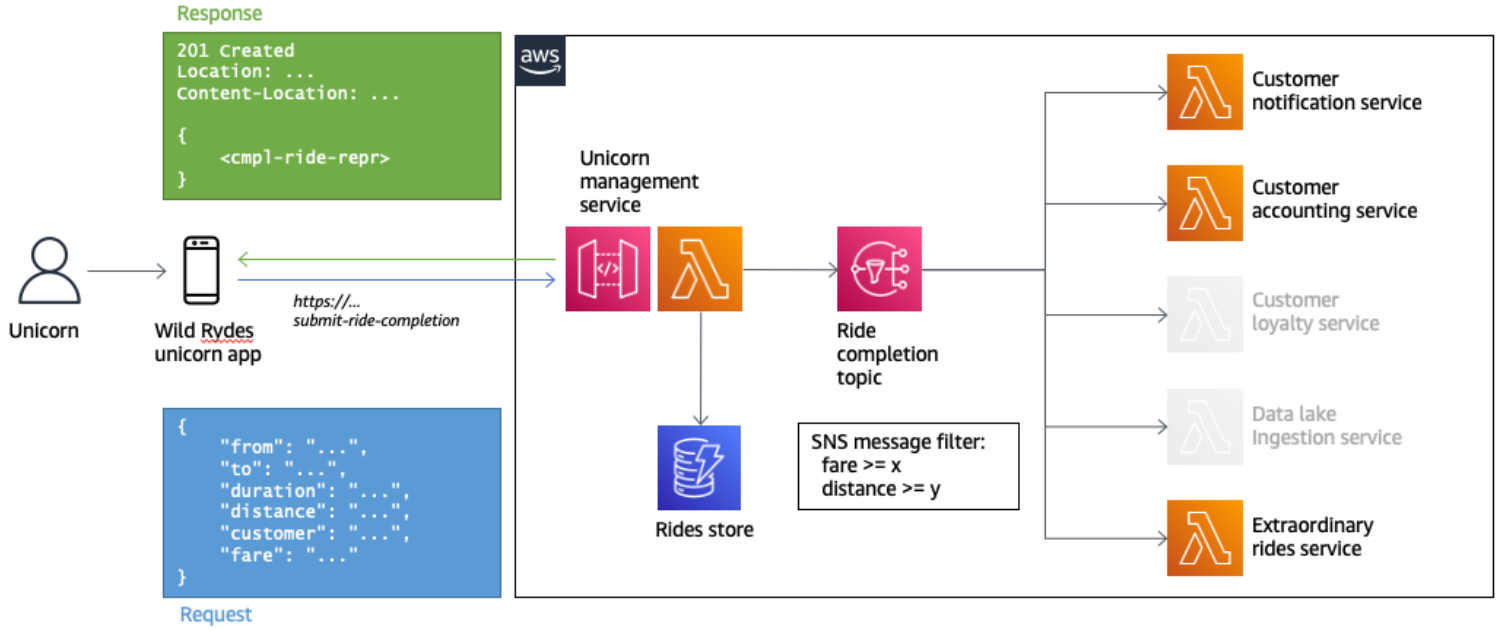
EventBridge Event JSON Object


```
{
 "version": "0",
 "id": "6a7e8feb-b491-4cf7-a9f1-bf3703467718",
 "detail-type": "EC2 Instance State-change Notification",
 "source": "aws.ec2",
 "account": "111122223333",
 "time": "2017-12-22T18:43:48Z",
 "region": "us-west-1",
 "resources": [
  "arn:aws:ec2:us-west-1:123456789012:instance/i-
1234567890abcdef0"
 ],
 "detail": {
  "instance-id": " i-1234567890abcdef0",
  "state": "terminated"
 }
}
```

Fan-Out & Message Filtering with Publish/Subscribe Pattern

- **Async Communication** for performing one-to-many and publish/subscribe mechanisms.
- Client service **publish a message** and it **consumes** from **several microservices** which's are subscribing this message on the message broker system.
- **Decouples Messaging** between services, building loosely-coupled architectures.
- Using in **event-driven** architectures
- **Publishes** an event something happen
- Price change in a product microservice
- **Subscribed** from SC microservice

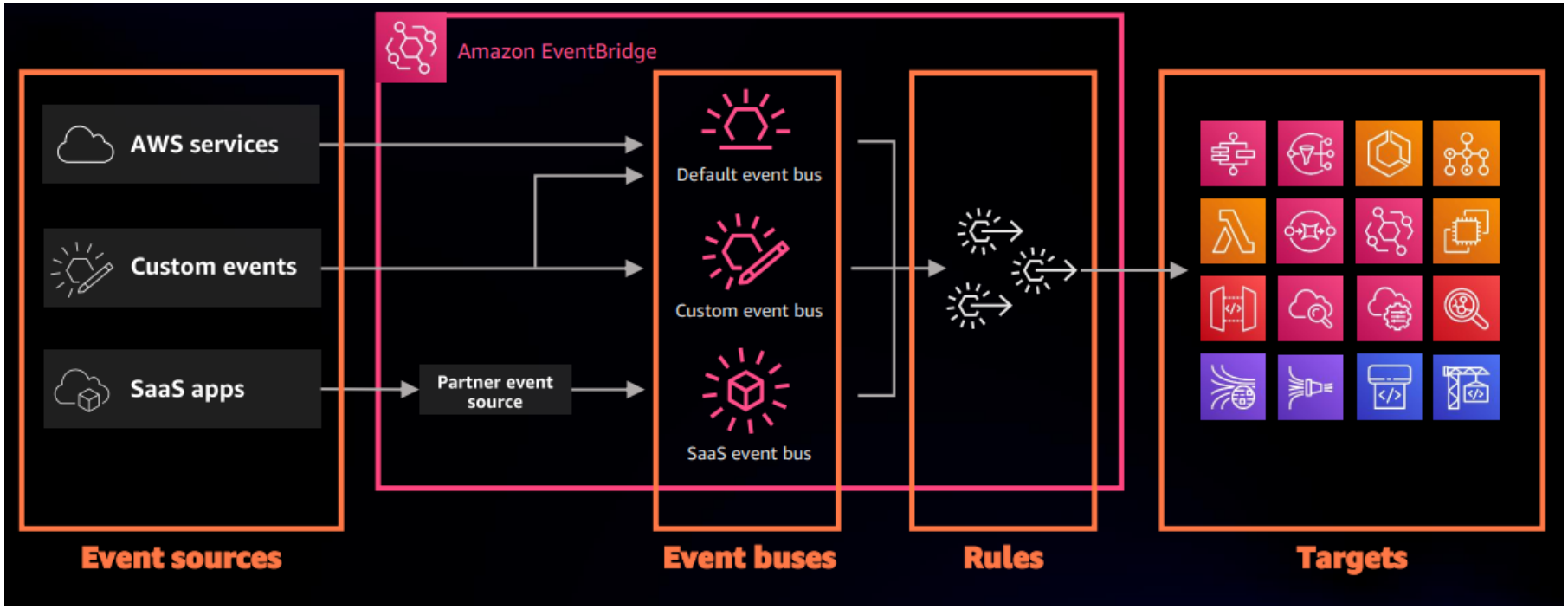
  to update basket price



https://async-messaging.workshop.aws/fan-out-and-message-filtering.html

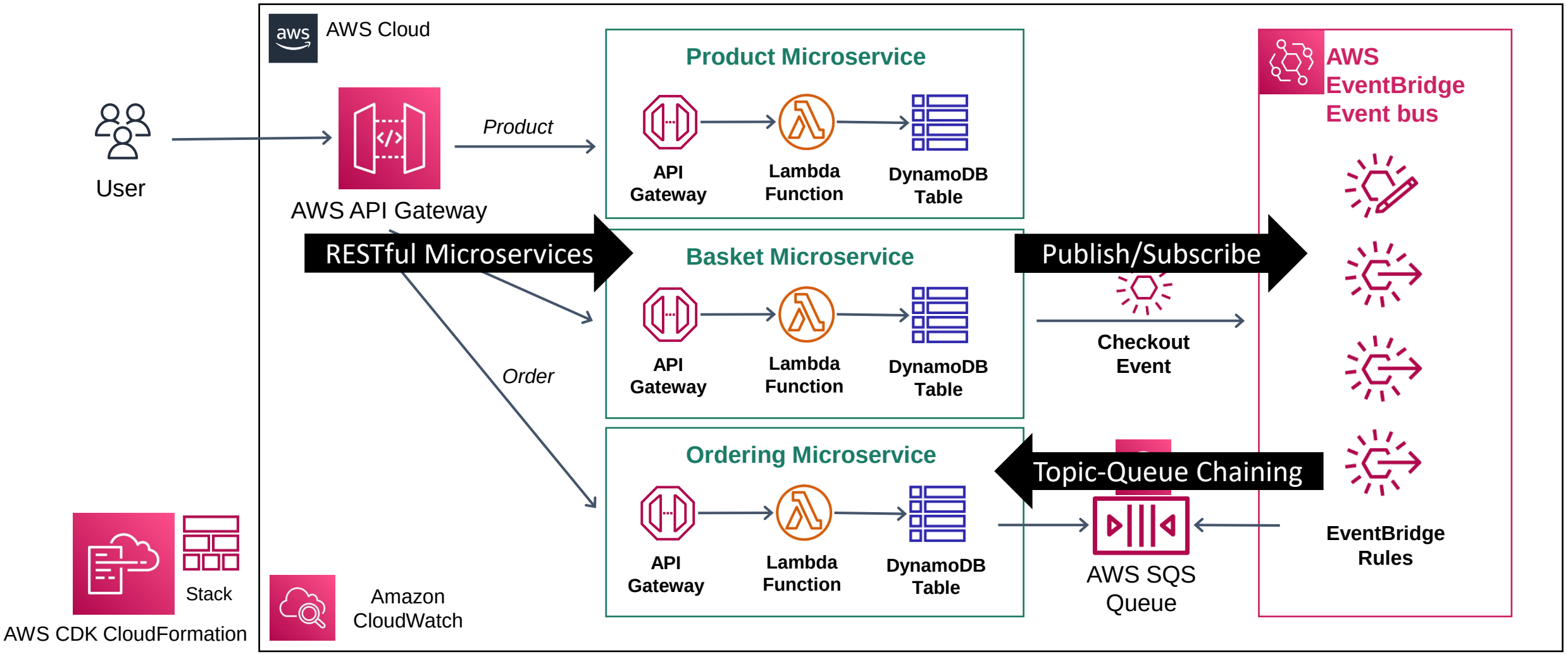
Publish/Subscribe Pattern with Amazon EventBridge

- **Amazon EventBridge** for Event-Driven asynchronous Communication between Microservices
- Event Sources
- Event Buses
- Rules
- Targets



https://da-public-assets.s3.amazonaws.com/serverlessland/pdf/2021+-+Serverlesspresso+exhibit+-+PDF.pdf

Serverless Patterns for Microservices



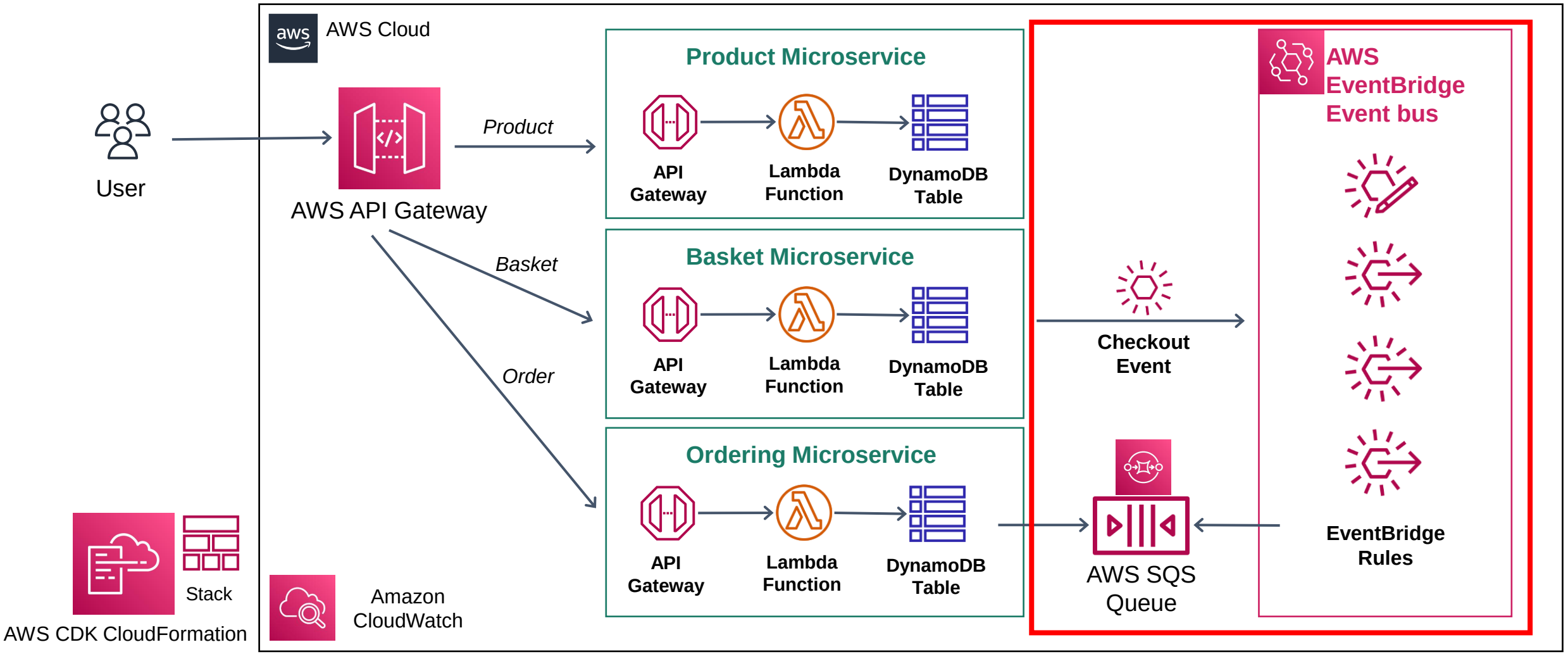
DEMO - Amazon EventBridge Walkthrough

→ Amazon EventBridge Walkthrough with AWS Management Console

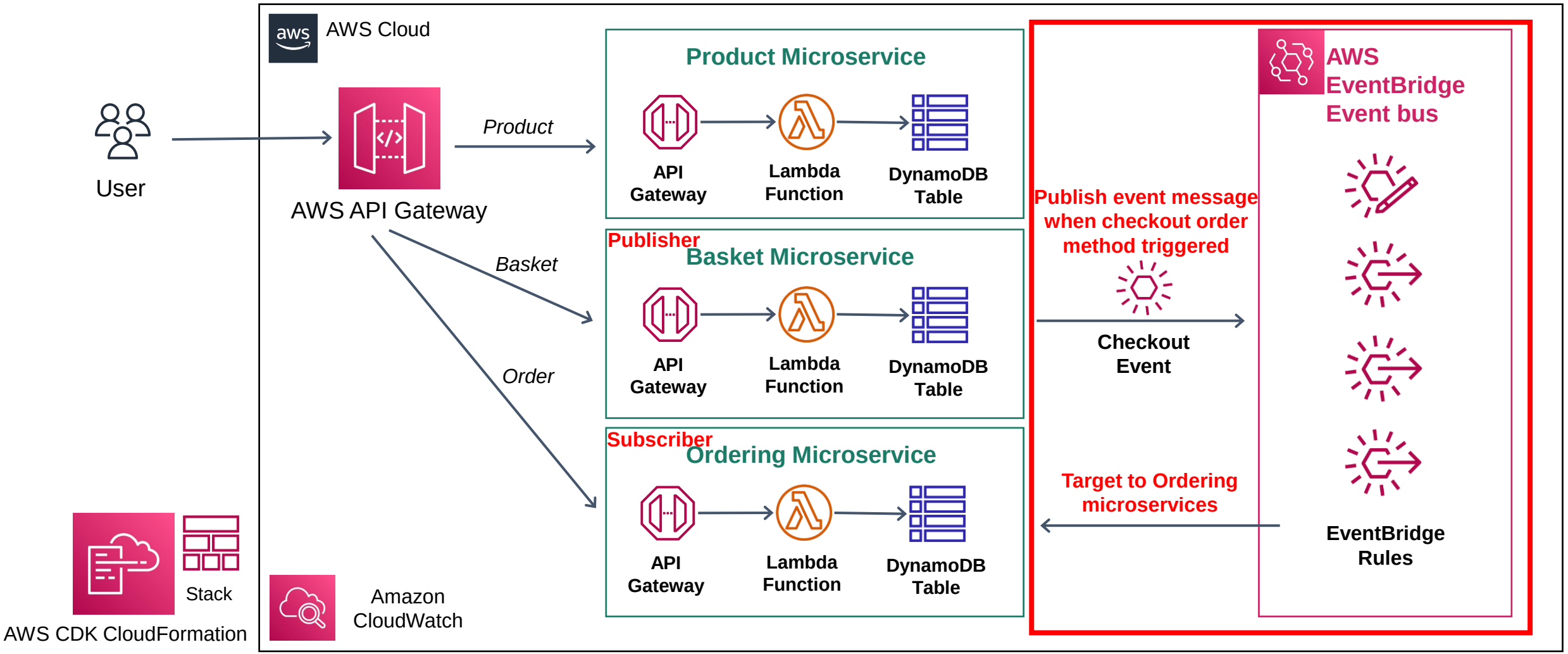
Creating AWS EventBridge Eventbus Infrastructure

→ Creating AWS EventBridge Eventbus Infrastructure with AWS CDK - Checkout Basket Async Flow

Serverless E-Commerce Microservices



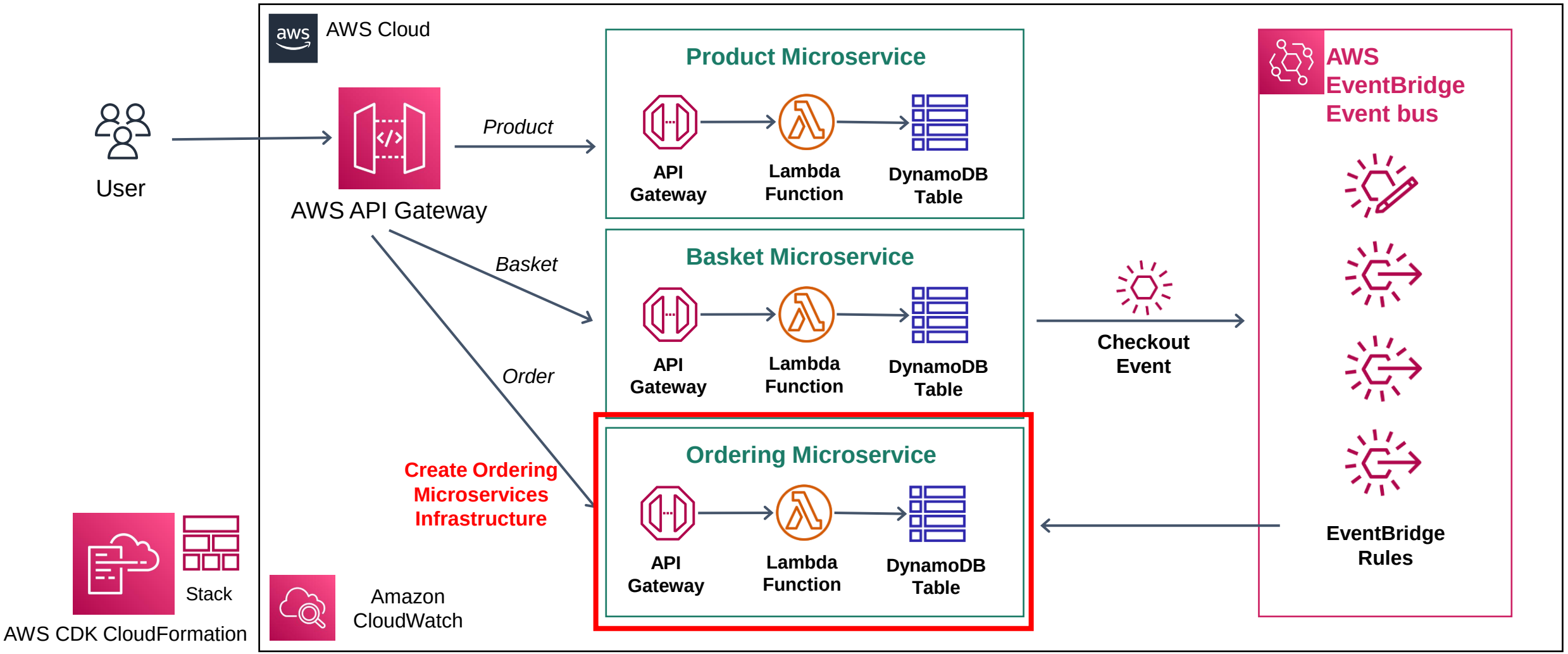
Serverless E-Commerce Microservices



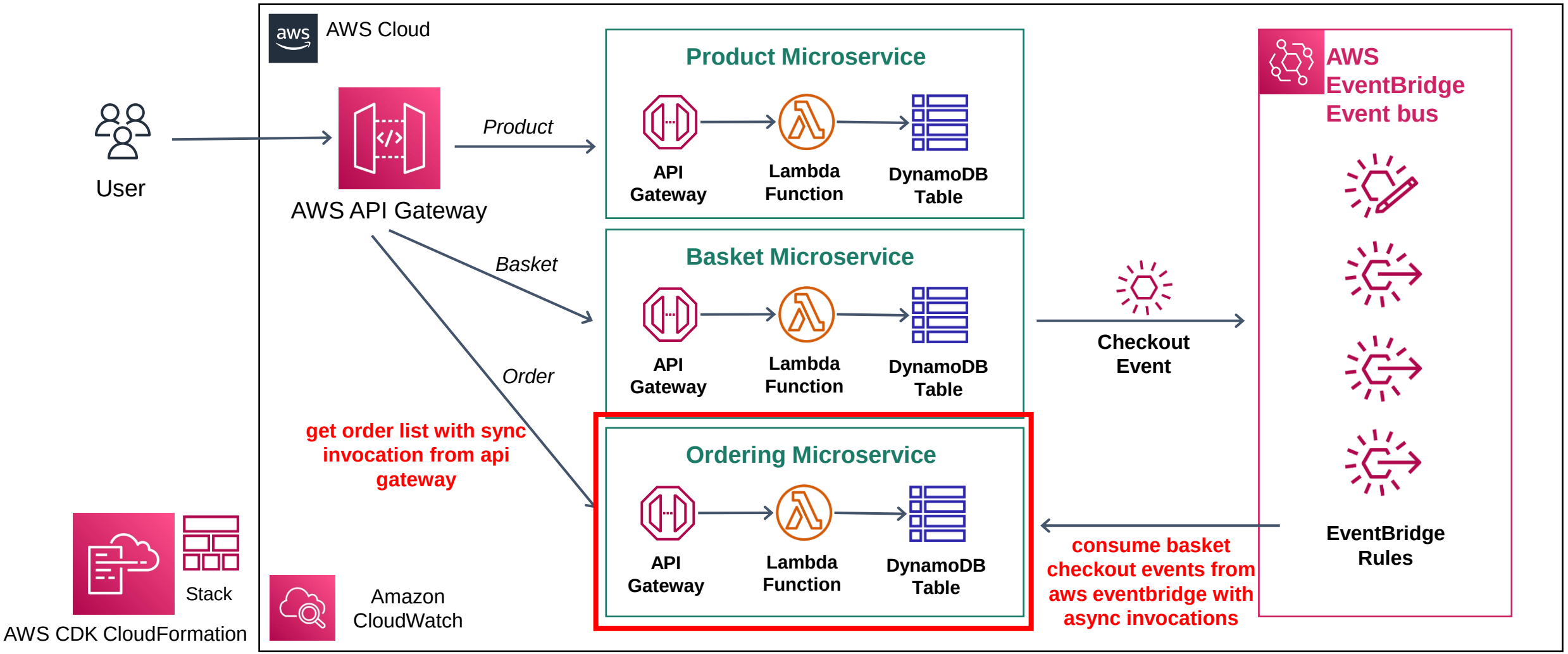
Creating Ordering Microservices Infrastructure

Creating Ordering Microservices Infrastructure with AWS CDK - DynamoDb - Lambda - Api Gateway

Serverless E-Commerce Microservices



Serverless E-Commerce Microservices



Clean up Resources

- **Amazon CTO Werner Vogels**
  Turn off the lights before leaving the room
- If you leave from computer and shut the light off, that means you also **shut down the AWS Resources**
- Think that we are going to prepare **demo session** with **your customer**
- Follow the best practices; **clear all resources** after you finished the work at that day.
- **Don’t worry**, when you come back tomorrow, we can instantly come the same situation with the power of **AWS CDK.**

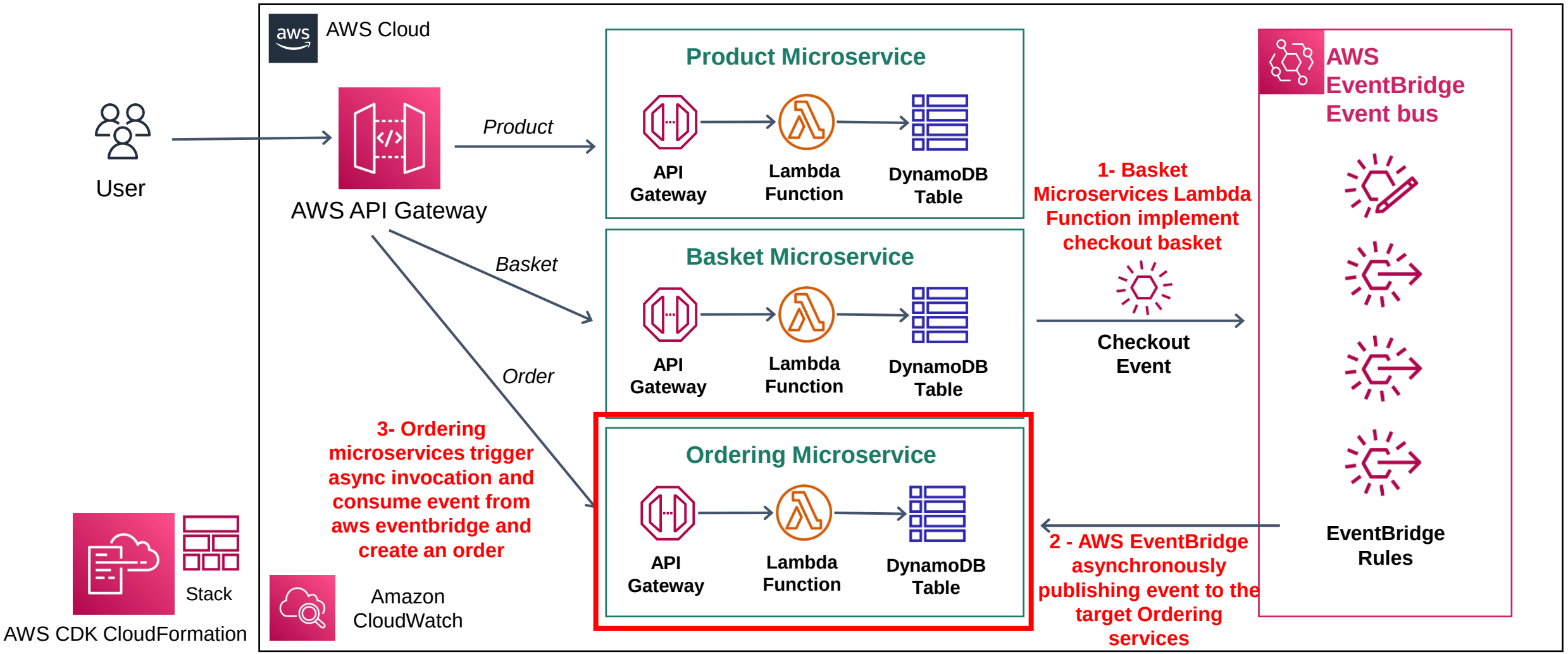
https://www.youtube.com/watch?v=8_Xs8Ik0h1w

Developing Checkout Basket for Send Event to AWS EventBridge

→ Developing Checkout Basket for Send Event to AWS EventBridge From Basket Microservice with AWS SDK

Checkout basket use case is starting async communication between basket and ordering microservices with decoupling aws event bridge.

Serverless E-Commerce Microservices



Clean up Resources

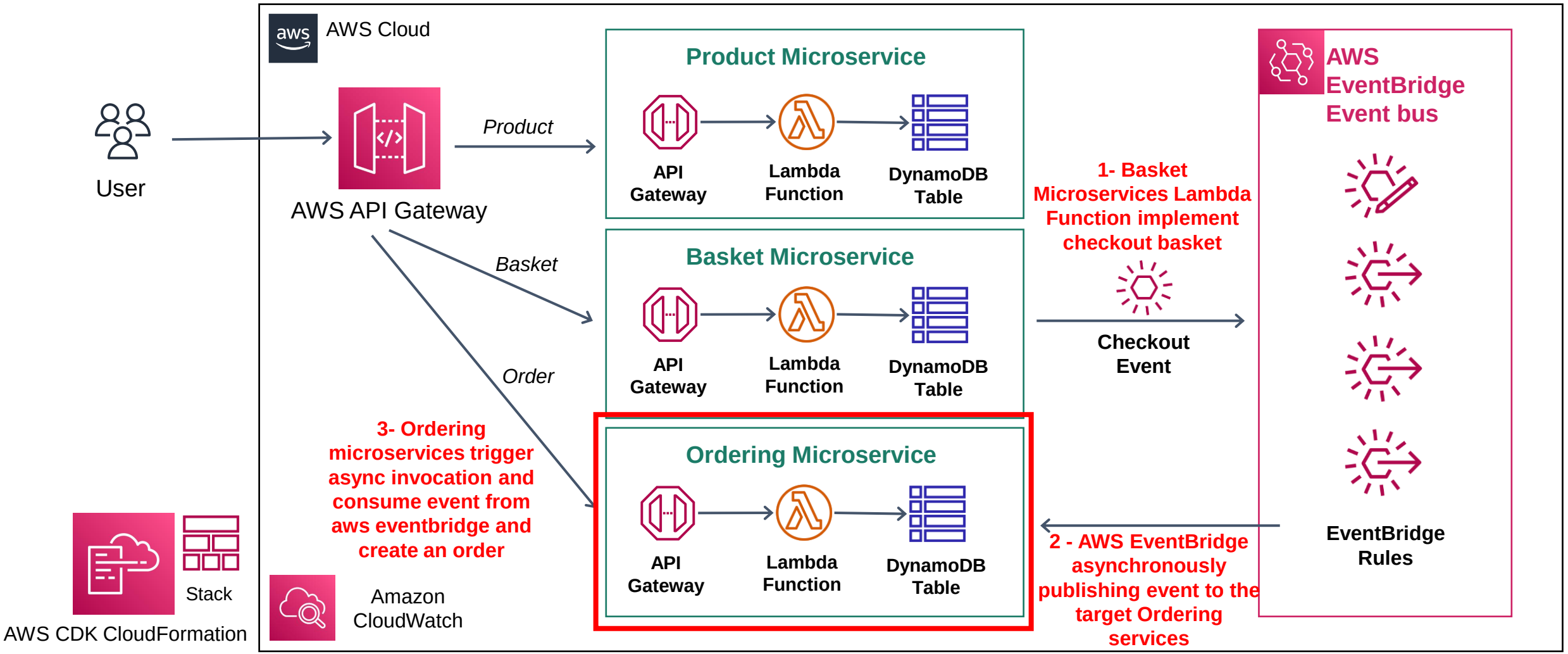
- **Amazon CTO Werner Vogels**
  Turn off the lights before leaving the room
- If you leave from computer and shut the light off, that means you also **shut down the AWS Resources**
- Think that we are going to prepare **demo session** with **your customer**
- Follow the best practices; **clear all resources** after you finished the work at that day.
- **Don’t worry**, when you come back tomorrow, we can instantly come the same situation with the power of **AWS CDK.**

https://www.youtube.com/watch?v=8_Xs8Ik0h1w

Developing Ordering Microservice Consume CheckoutBasket Event

Developing Ordering Microservice Consume CheckoutBasket Event from AWS EventBridge Async Invocation with AWS SDK

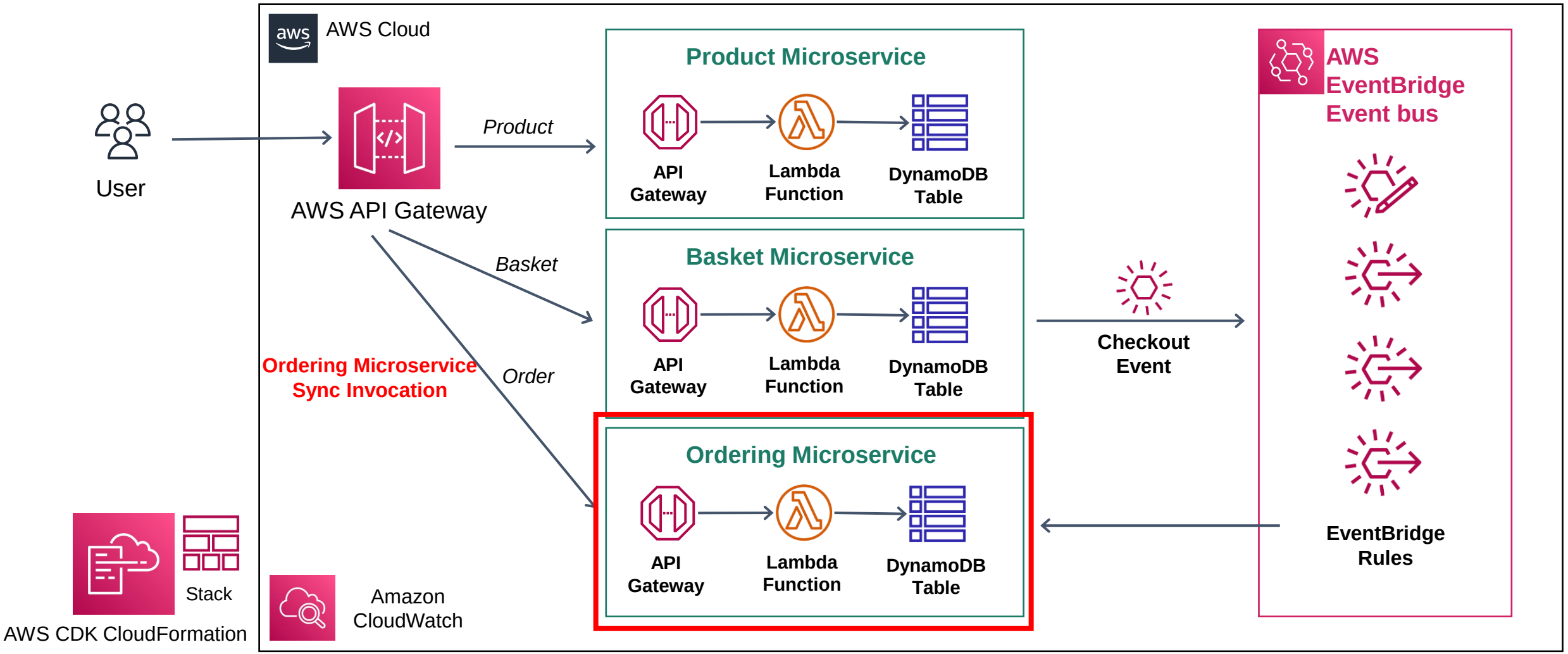
Serverless E-Commerce Microservices



Developing Ordering Microservice Sync Invocation

→ Developing Ordering Microservice Sync Invocation from Api Gateway with AWS SDK

Serverless E-Commerce Microservices



E2E Testing Basket and Ordering Microservices

- E2E Testing Basket and Ordering Microservices Sync & Async Invocations
- Testing Basket Microservices CheckoutBasket EventBridge Async Flow
- Testing Ordering Microservice Api Gateway Sync Flow

Clean up Resources

- **Amazon CTO Werner Vogels**
  Turn off the lights before leaving the room
- If you leave from computer and shut the light off, that means you also **shut down the AWS Resources**
- Think that we are going to prepare **demo session** with **your customer**
- Follow the best practices; **clear all resources** after you finished the work at that day.
- **Don’t worry**, when you come back tomorrow, we can instantly come the same situation with the power of **AWS CDK.**

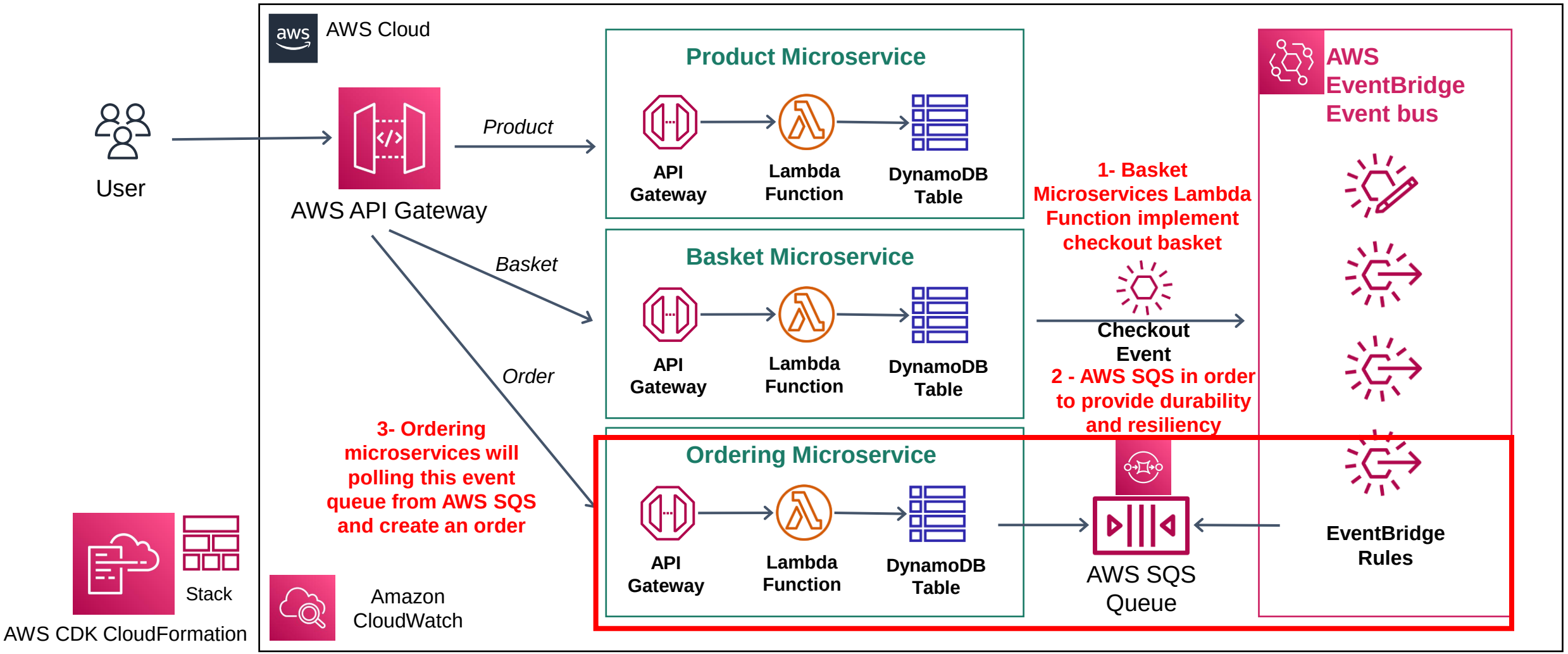
https://www.youtube.com/watch?v=8_Xs8Ik0h1w

AWS SQS for Event-Driven Architecture with Queues

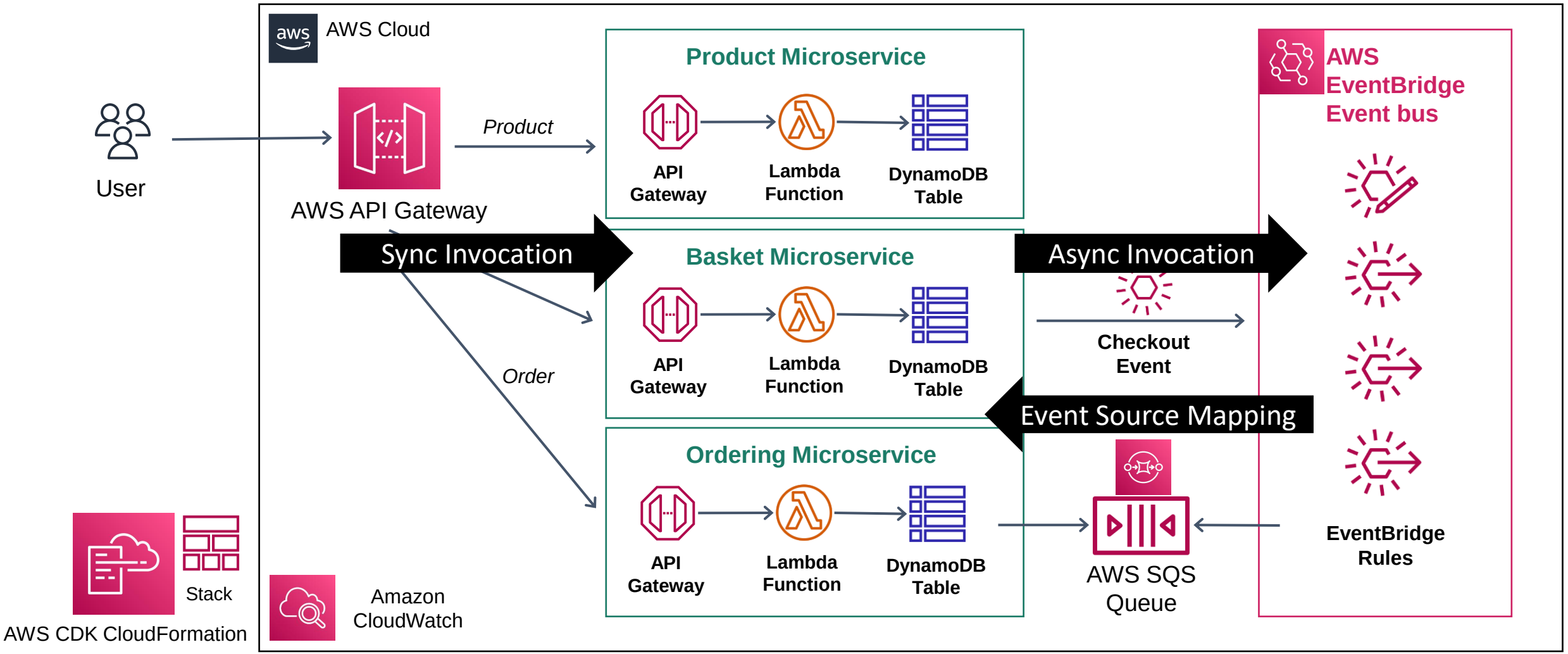
AWS SQS for Event-Driven Architecture with Queues

How we can use SQS as a Queue for implementing queue patterns in ordering microservices with using Event-Driven Architecture best practices.

AWS Serverless E-Commerce Microservices



AWS Serverless Microservices with AWS Lambda Invocation Types



What is Amazon SQS ?

Amazon SQS

- Amazon SQS stands for **Simple Queue Service** is **fully managed message queues** for microservices, distributed systems, and serverless applications.
- Enables you to **decouple** and **scale microservices**, distributed systems, and serverless applications.
- **Eliminates** the **complexity** and overhead associated with managing and operating **message-oriented middleware.**
- **Send**, **store**, and **receive messages** between software components at any volume.
- Two types of message queues.
  - **Standard queues** offer maximum throughput, best-effort ordering, and at-least-once delivery.
  - **FIFO queues** are designed to guarantee that messages are processed exactly once, in the exact order that they are sent.
- **Integrate** and **decouple** distributed software systems and components.
- Provides a **generic web services API** that you can access using any programming language that the **AWS SDK** supports.

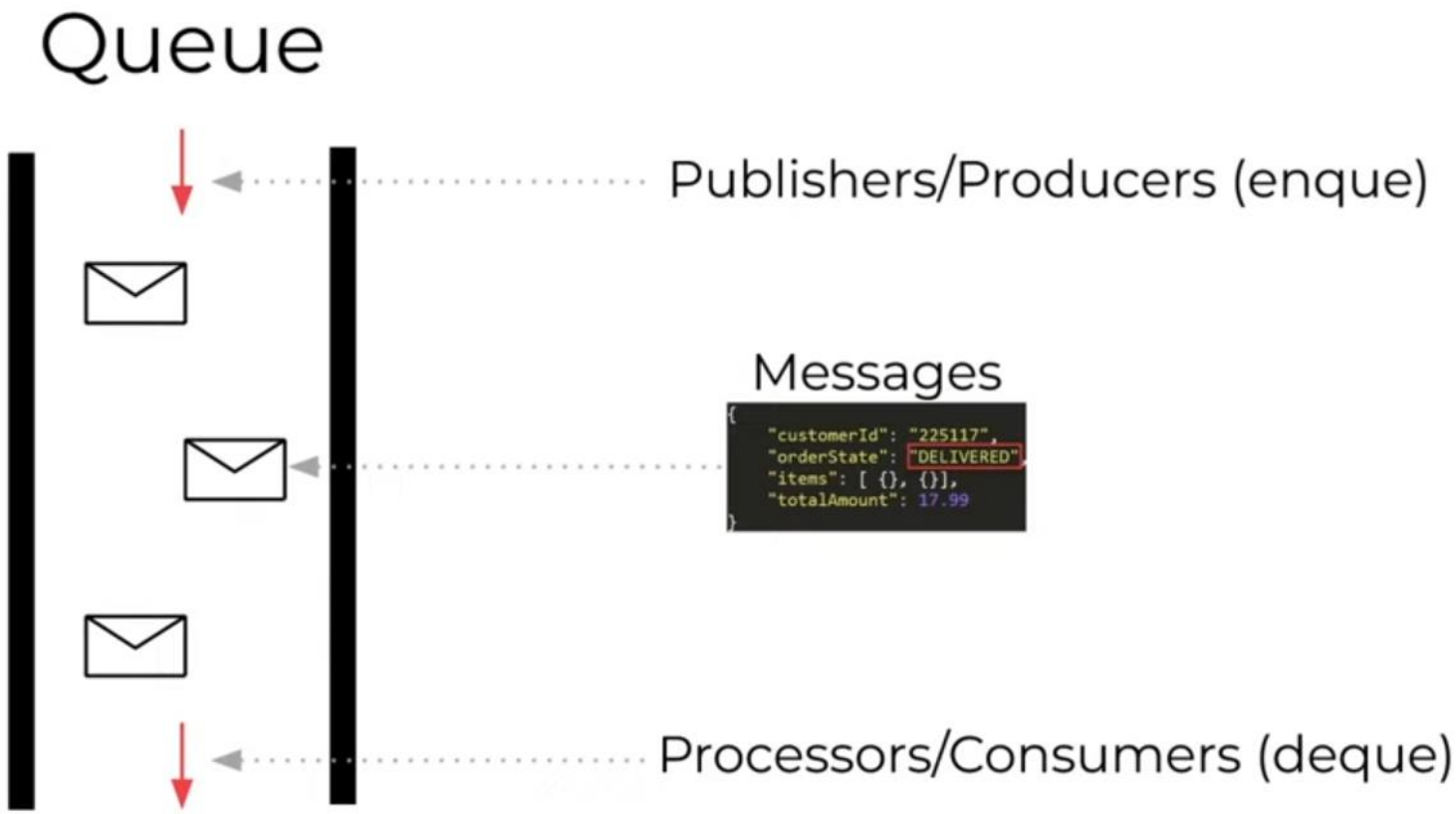
Benefits of Amazon SQS

Amazon SQS

- **Eliminate administrative overhead**
  AWS manages all ongoing operations and underlying infrastructure needed to provide a highly available and scalable message queuing service. SQS queues are dynamically created and scale automatically.
- **Durability and Reliability deliver messages**
  Amazon SQS stores them on multiple servers. Standard queues support at-least-once message delivery, and FIFO queues support exactly-once message processing. SQS locks your messages during processing, so that multiple producers can send and multiple consumers can receive messages at the same time.
- **Scalability and Availability and cost-effectively**
  SQS scales elastically with your application so you don't have to worry about capacity planning and pre-provisioning. There is no limit to the number of messages per queue, and standard queues provide nearly unlimited throughput.
- **Security - Keep sensitive data secure**
  Amazon SQS to exchange sensitive data between applications using server-side encryption (SSE) to encrypt each message body.

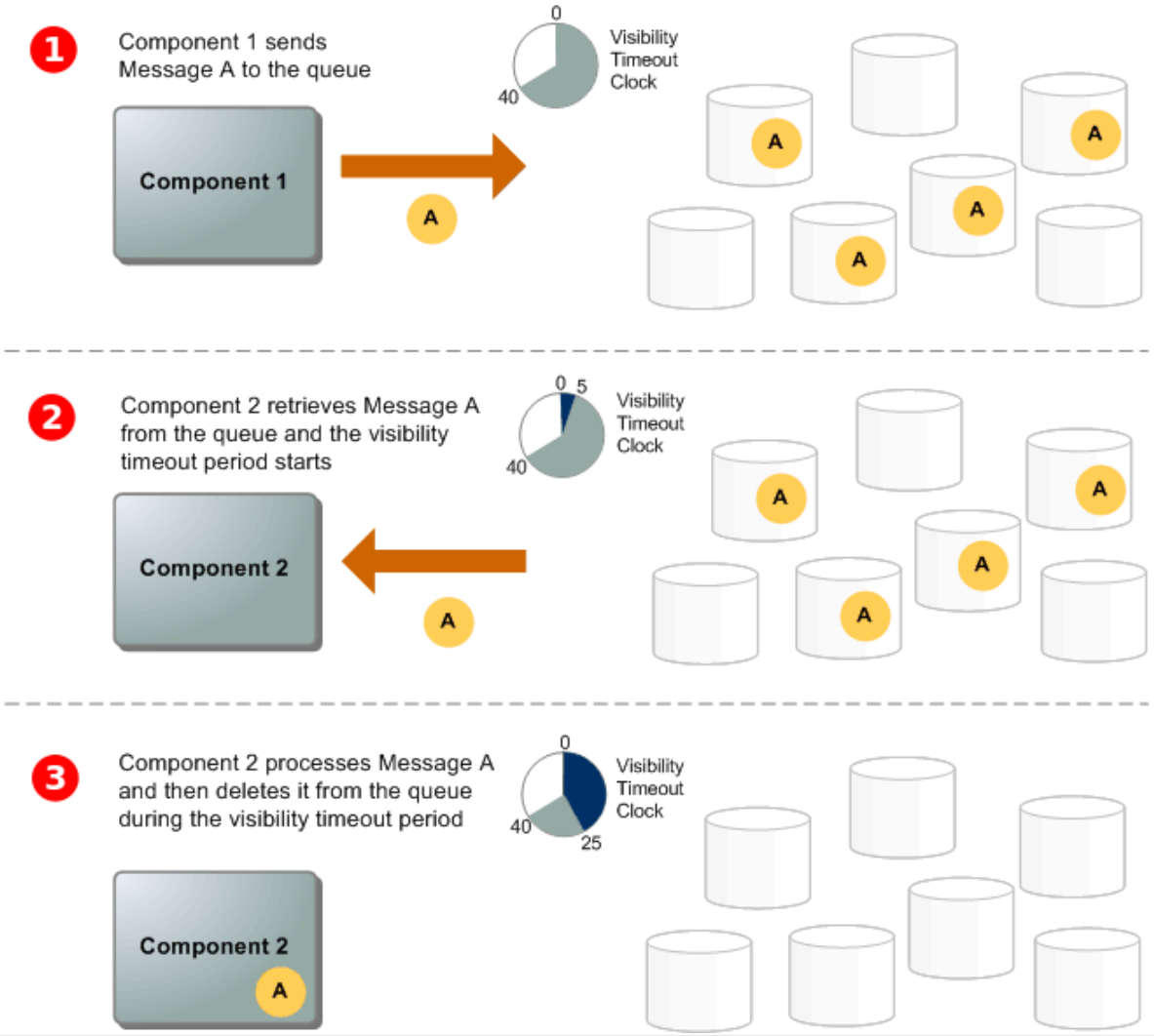
Amazon SQS architecture and How SQS works

Amazon SQS



https://www.youtube.com/watch?v=CyYZ3adwboc&t=152s

The lifecycle of an Amazon SQS message



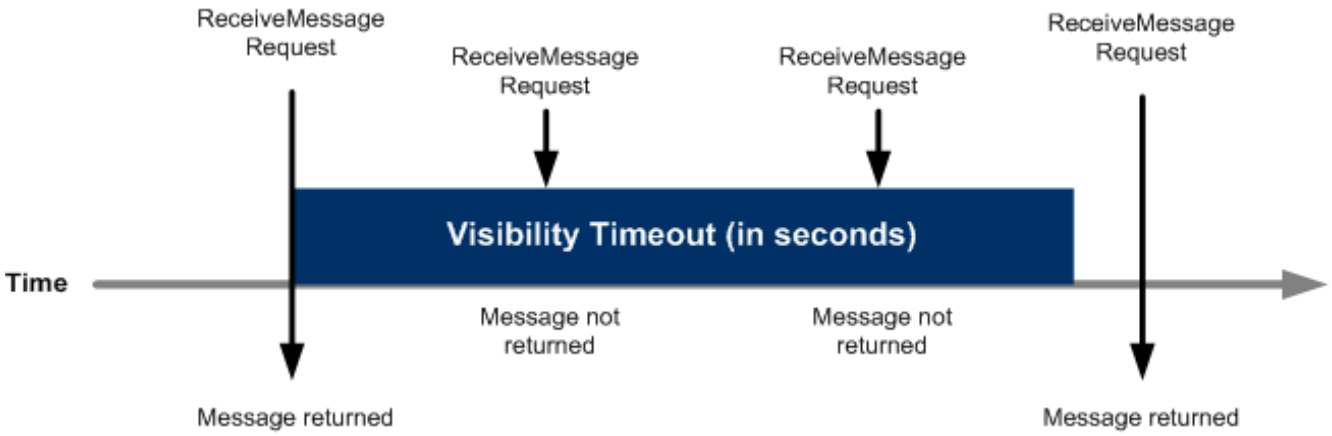
Amazon SQS

https://docs.aws.amazon.com/AWSSimpleQueueService/latest/SQSDeveloperGuide/sqs-basic-architecture.html

Amazon SQS Visibility Timeout

Amazon SQS

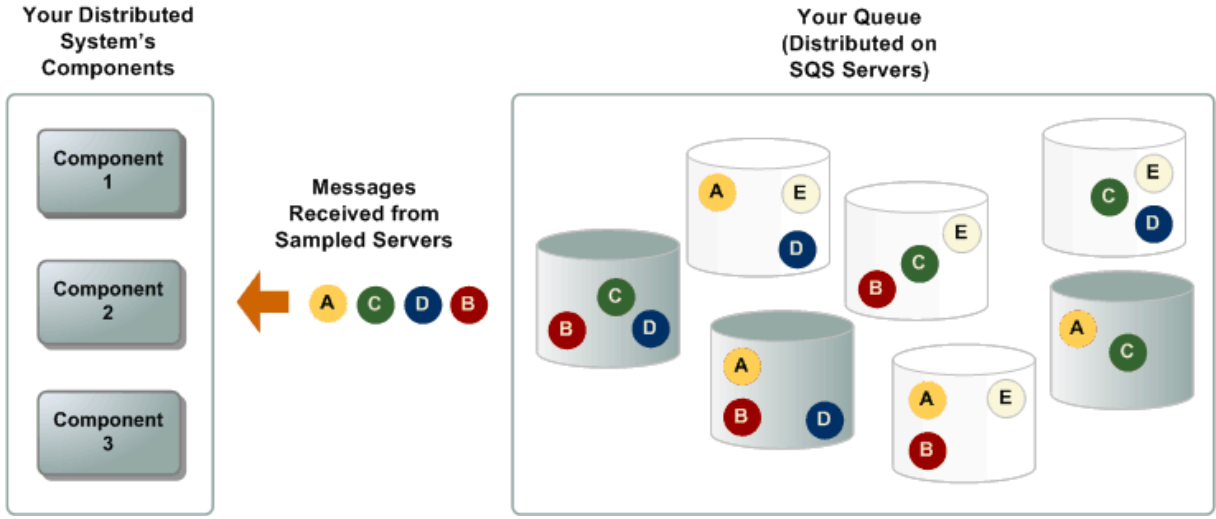
- When a **consumer receives** and **processes** a message from a **queue**, the message remains in the queue.
- the consumer **must delete the message** from the queue after receiving and processing it.
- **Visibility timeout;** a period of time during which Amazon SQS prevents other consumers from receiving and processing the message.
- The **default visibility timeout** for a message is 30 seconds.
- Configuring **visibility timeout** for a **queue** using the console.



https://docs.aws.amazon.com/AWSSimpleQueueService/latest/SQSDeveloperGuide/sqs-visibility-timeout.html

Amazon SQS Short and Long polling

- By default, queues use **short polling**.
- **Short polling**, the **ReceiveMessage request** queries only a subset of the servers to find messages that are available to include in the response.
- **Long polling,** the **ReceiveMessage request** queries all of the servers for messages. Amazon SQS sends a response after it collects at least one available message.
- Consume messages using **short polling**, Amazon SQS samples **a subset** of its servers and returns messages from only those servers.
- When the **wait time** for the **ReceiveMessage** API action is greater than 0, long polling is in effect. Long polling helps **reduce** the **cost of using Amazon SQS** by eliminating the number of empty responses.



https://docs.aws.amazon.com/AWSSimpleQueueService/latest/SQSDeveloperGuide/sqs-short-and-long-polling.html

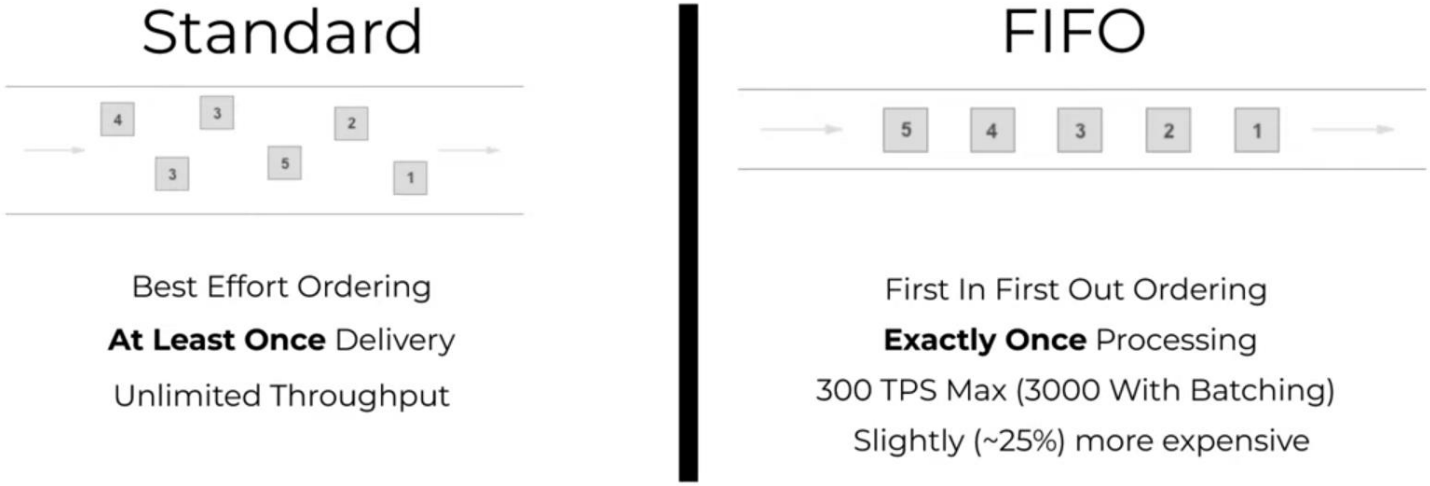
Amazon SQS Dead-letter Queues

- Amazon SQS supports **dead-letter queues** (**DLQ**)
- Messages **can't be processed** because of a variety of possible issues.
- **Example**; User places a web order with a particular product ID, but the product ID is deleted. Code fails and displays an error, and the message with the order request is sent to a **dead-letter queue.**
- The main task of a dead-letter queue is to **handle the lifecycle of unconsumed messages**.
- A **dead-letter queue** lets you set **aside** and **isolate** messages that can't be processed correctly to determine why their processing didn't succeed.
- Setting up a dead-letter queue
  - Configure an alarm for any messages moved to a dead-letter queue.
  - Examine logs for exceptions that might have caused messages to be moved to a dead-letter queue.
  - Analyze the contents of messages moved to a dead-letter queue.
  - Determine whether you have given your consumer sufficient time to process messages.

SQS Queue types - Standard Queues and FIFO Queues

Amazon SQS

- SQS Provides two types of Message Queue
  - Standard Queue
  - FIFO Queue
- **Standard Queue** is the default queue type offered by AWS SQS. This provides unlimited throughput, best effort ordering and at least once message delivery.
- **FIFO Queue** is simply means that messages will be ordered in the queue and first message to arrive in the queue will be first to leave the queue.



https://aws.amazon.com/sqs/features/

SQS Queue types - Standard Queues and FIFO Queues

Amazon SQS

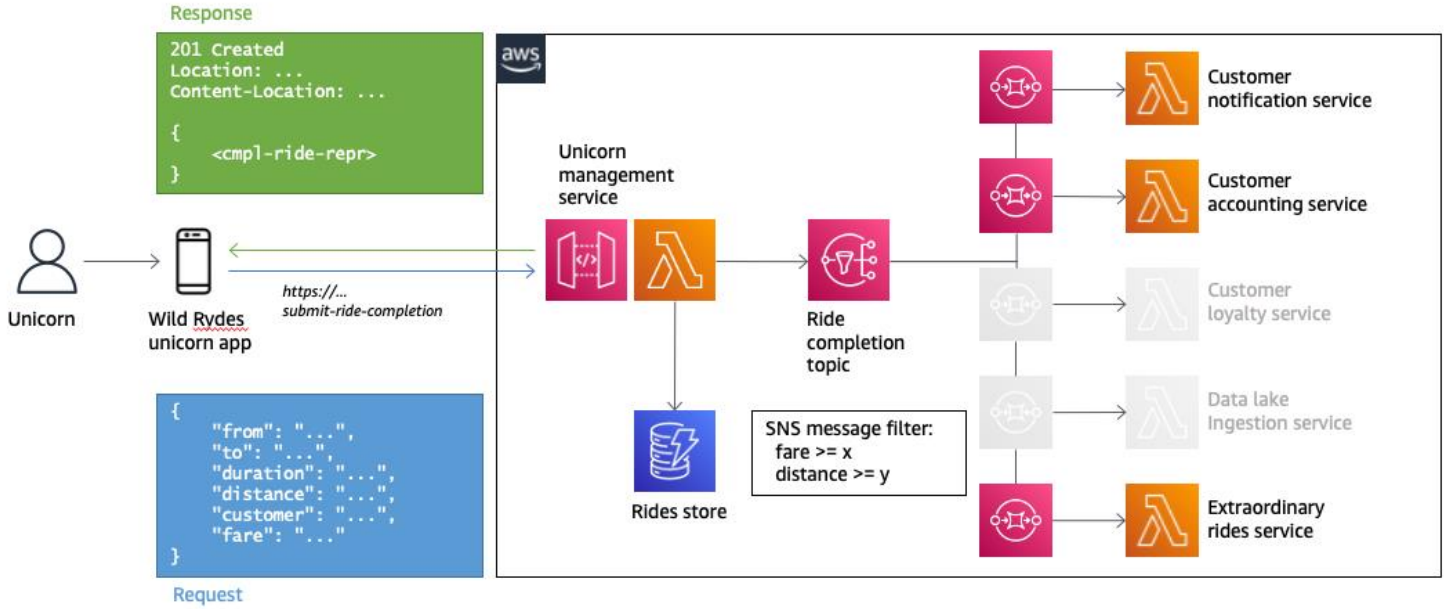
- **Message Ordering**
  SQS Standard queues provide best-effort ordering. Occasionally more than one copy of a message might be delivered out of order. FIFO queues offers first-in-first-out delivery.
- **Delivery**
  Standard Queue grantees at least once delivery but sometimes duplicates or more then one copy of a message can be deliver. FIFO queues ensure a message is delivered exactly once.
- **Throughput**
  Standard queues offers unlimited throughput. FIFO queues on the other hand are limited to 300 transactions per second per API action.
- **Use cases**
  Use Standard queue as long as you are able to process duplicates and out of order messages. FIFO queues can be used when ordering of message is must and duplicates are not accepted at any cost.
- Standard queues provide at-least-once delivery, which means that each message is delivered at least once.
- FIFO queues provide exactly-once processing.

Topic-Queue Chaining & Load Balancing Pattern

- Use a **queue** that acts as a **buffer** between the service to a**void loss data** if the service to fail.
- **Services** can be **down** or **getting exception** or taken offline for maintenance, then **events** will be **loses**, **disappeared** and can't process after the subscriber service is up and running.
- Put **Amazon SQS** between **EventBridge** and **Ordering** microservices.
- Store this event messages into **SQS queue**

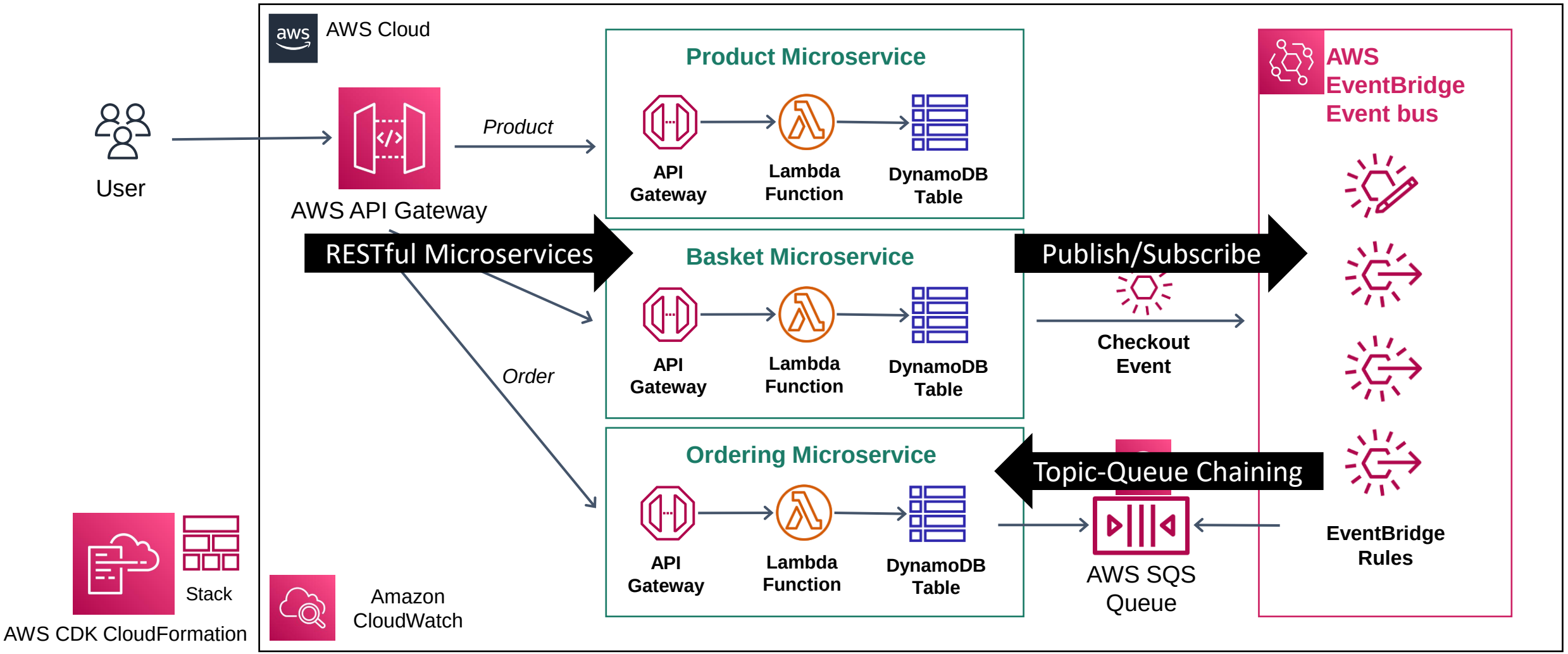
  with **durable** and **persistent** manner,

  no message will get lost
- **Queue** can act as a buffering **load balancer**



https://async-messaging.workshop.aws/fan-out-and-message-filtering.html

Serverless Patterns for Microservices



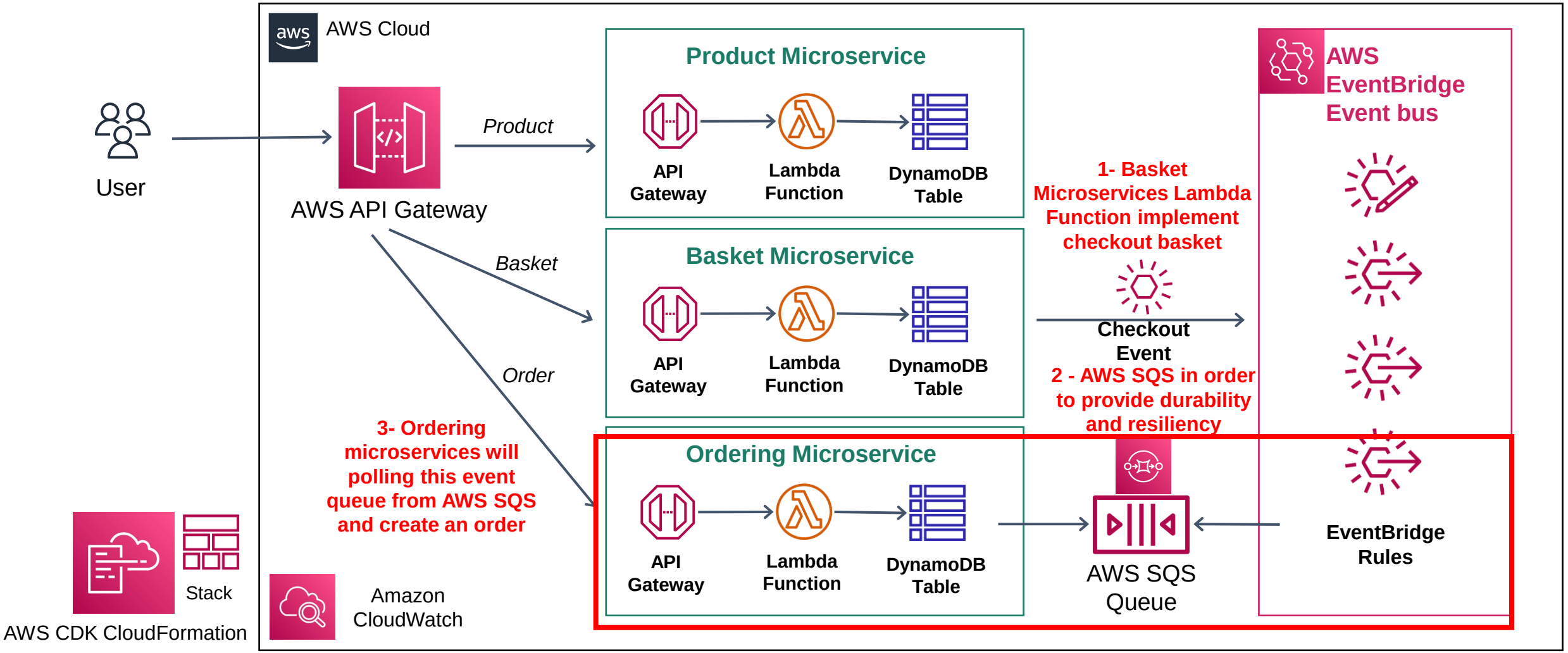
DEMO - Amazon SQS Walkthrough

→ Amazon SQS Walkthrough with AWS Management Console

Creating AWS SQS Queue Infrastructure with AWS CDK

→ Creating AWS SQS Queue Infrastructure with AWS CDK - Polling Checkout Basket Event with Event Source Mappings

AWS Serverless E-Commerce Microservices

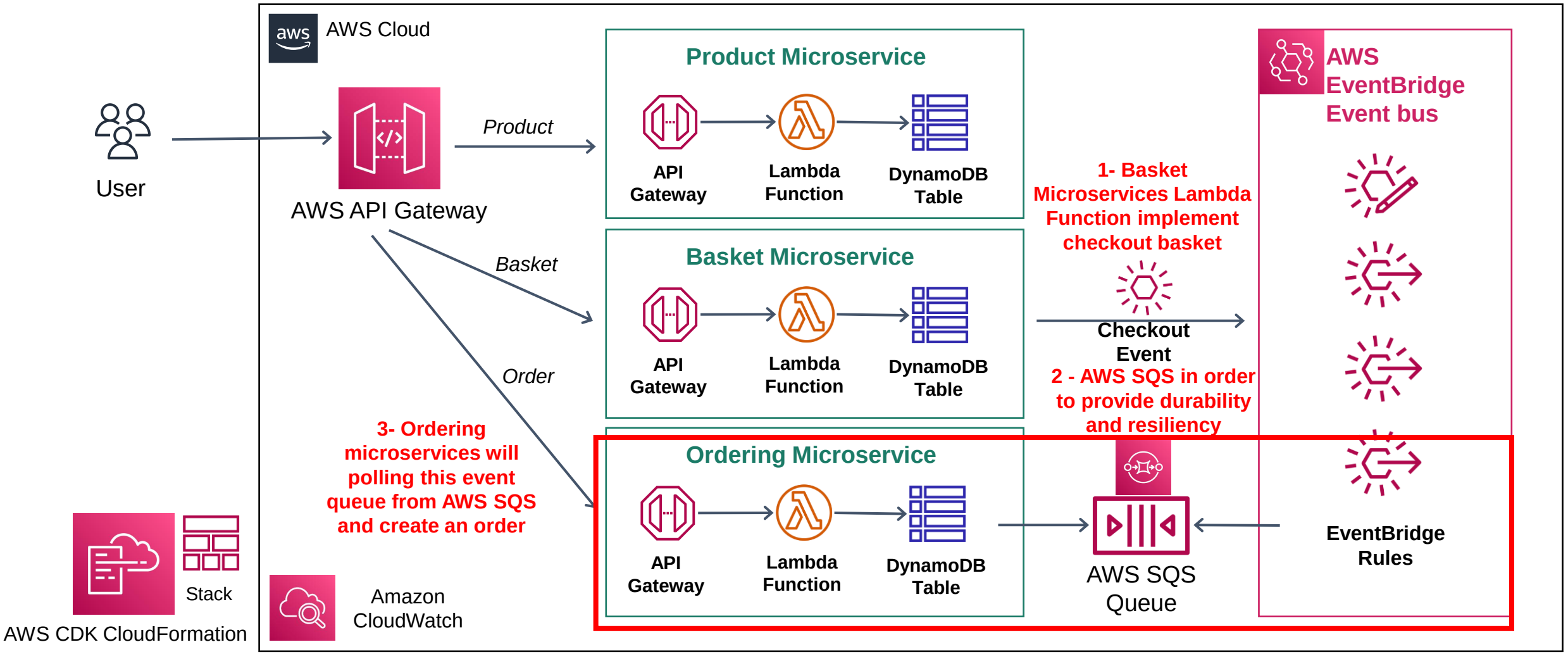


Part 1 - SQS Infrastructure as Code Development with CDK
Part 2 - Interaction with Ordering Microservices using AWS SDK

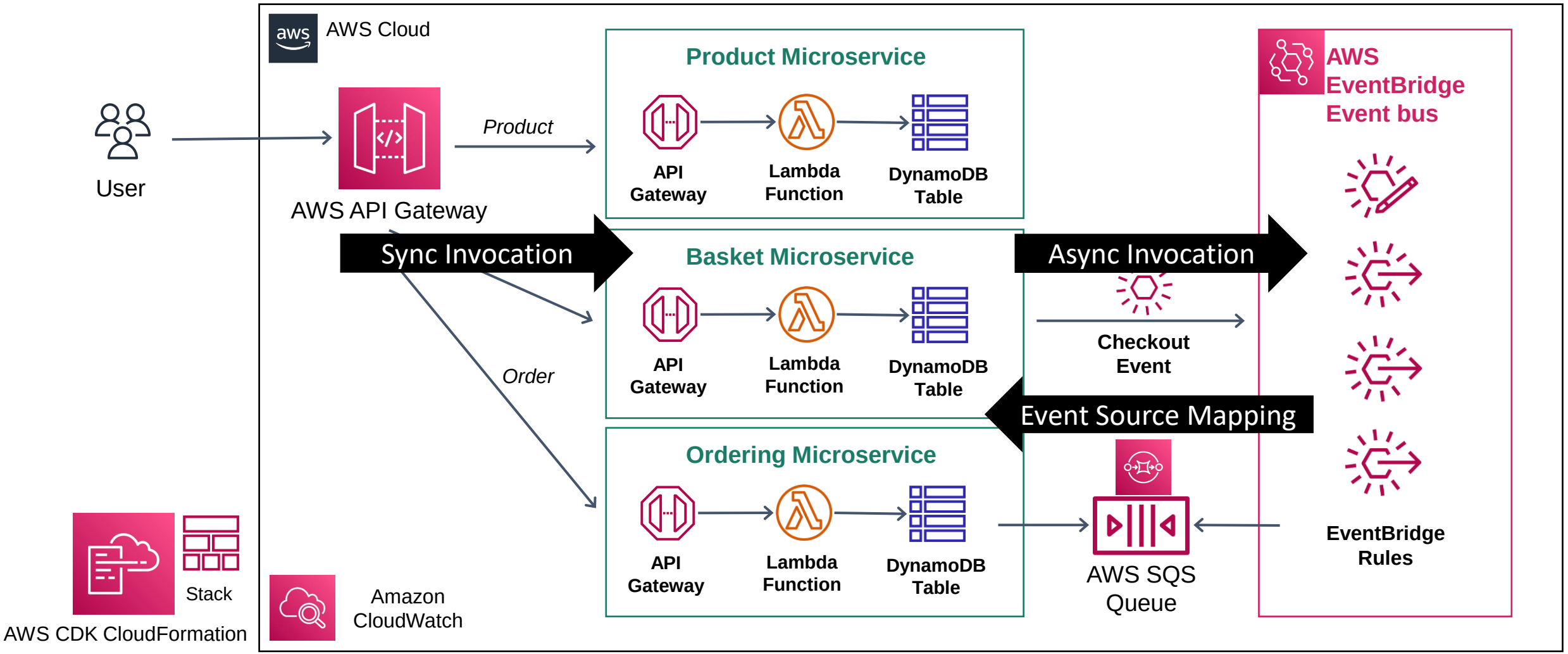
Developing AWS SQS Event Source Mapping Polling Invocation from Ordering Microservice

→ Developing AWS SQS Event Source Mapping Polling Invocation from Ordering Microservice Consume CheckoutBasket Event with AWS SDK

AWS Serverless E-Commerce Microservices



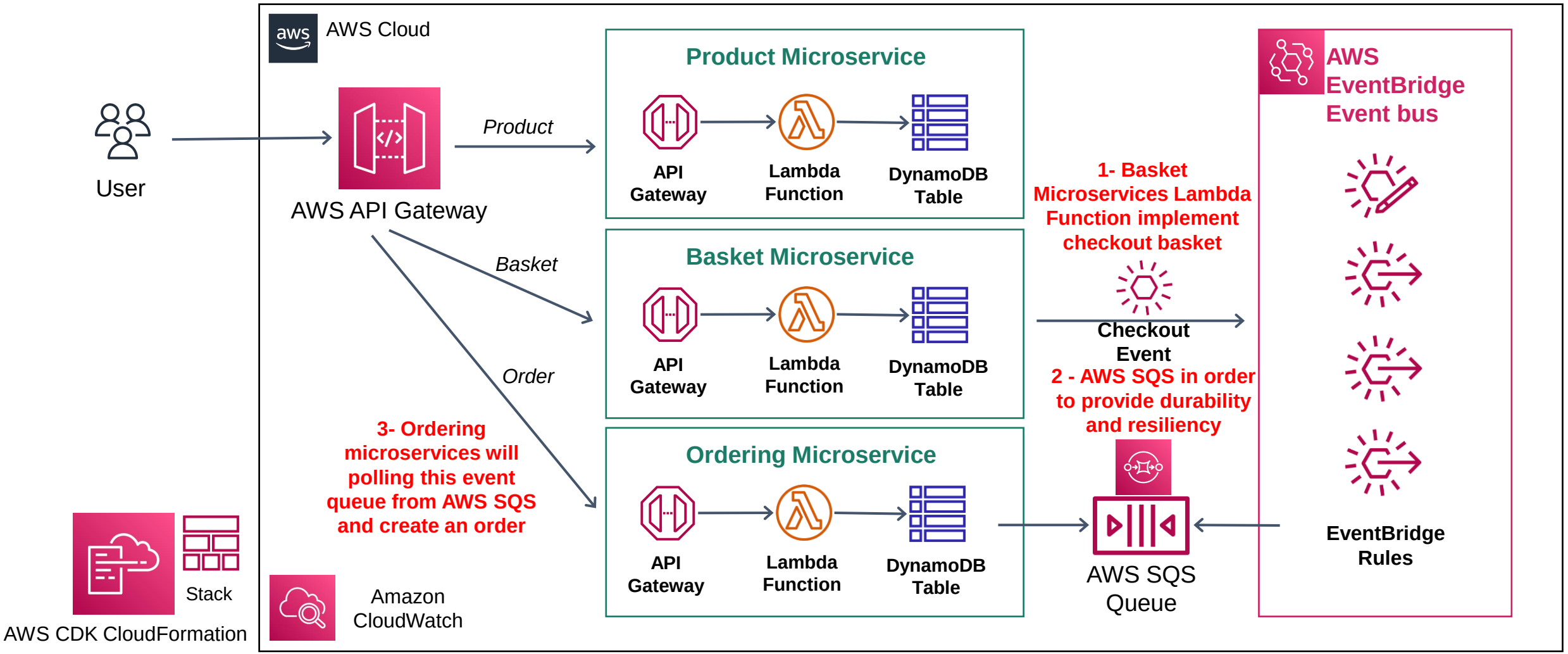
AWS Serverless Microservices with AWS Lambda Invocation Types



E2E Testing Basket and Ordering Microservices

- E2E Testing Basket and Ordering Microservices Event Source Mapping Polling Invocations
- Testing Basket Microservices CheckoutBasket EventBridge Async Flow
- Ordering Microservices Event Source Mapping AWS SQS Polling Invocations
- Testing Ordering Microservice Api Gateway Sync Flow

AWS Serverless E-Commerce Microservices



Clean up Resources

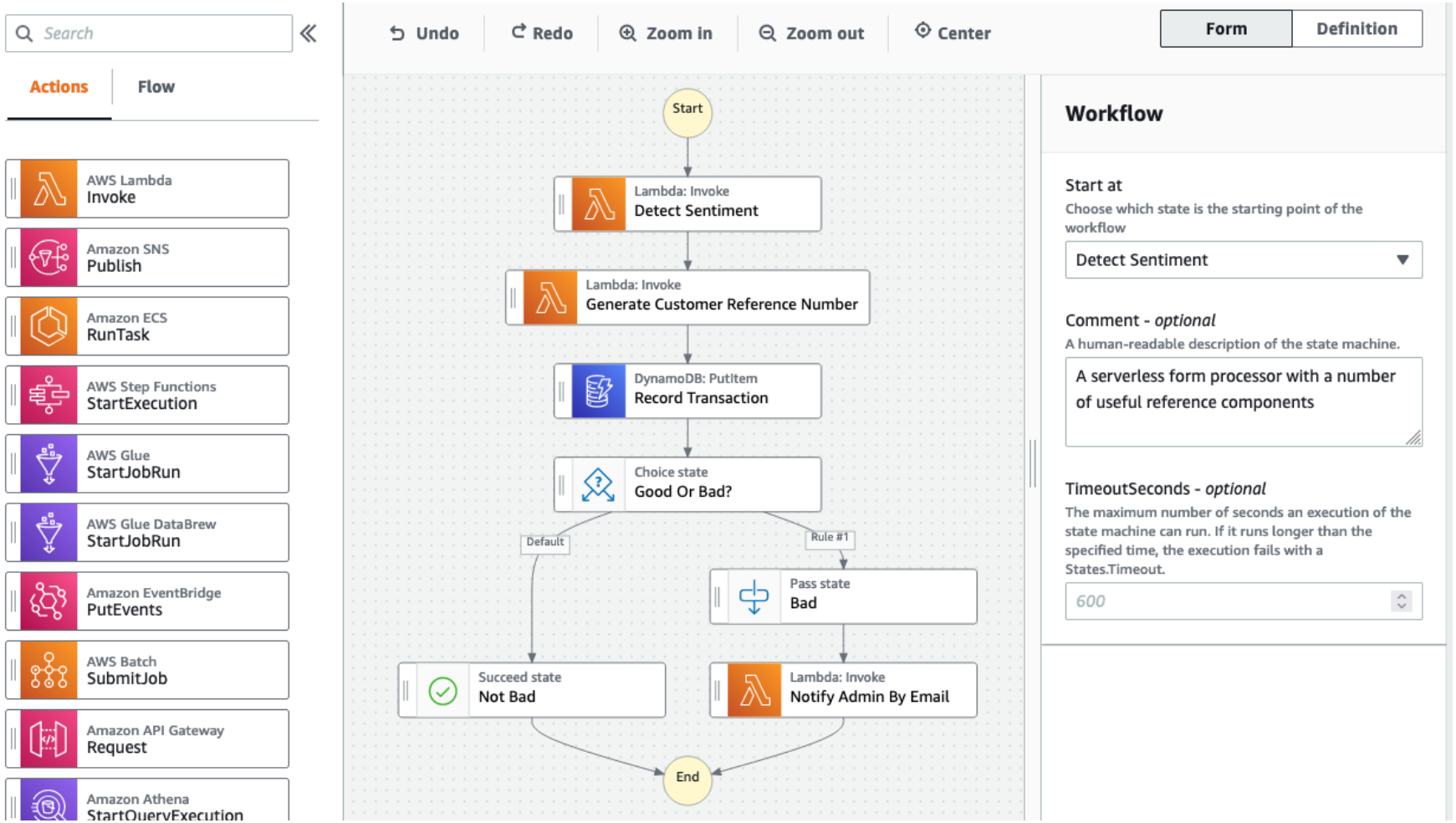
- **Amazon CTO Werner Vogels**
  Turn off the lights before leaving the room
- If you leave from computer and shut the light off, that means you also **shut down the AWS Resources**
- Think that we are going to prepare **demo session** with **your customer**
- Follow the best practices; **clear all resources** after you finished the work at that day.
- **Don’t worry**, when you come back tomorrow, we can instantly come the same situation with the power of **AWS CDK.**

https://www.youtube.com/watch?v=8_Xs8Ik0h1w

What's Next ?

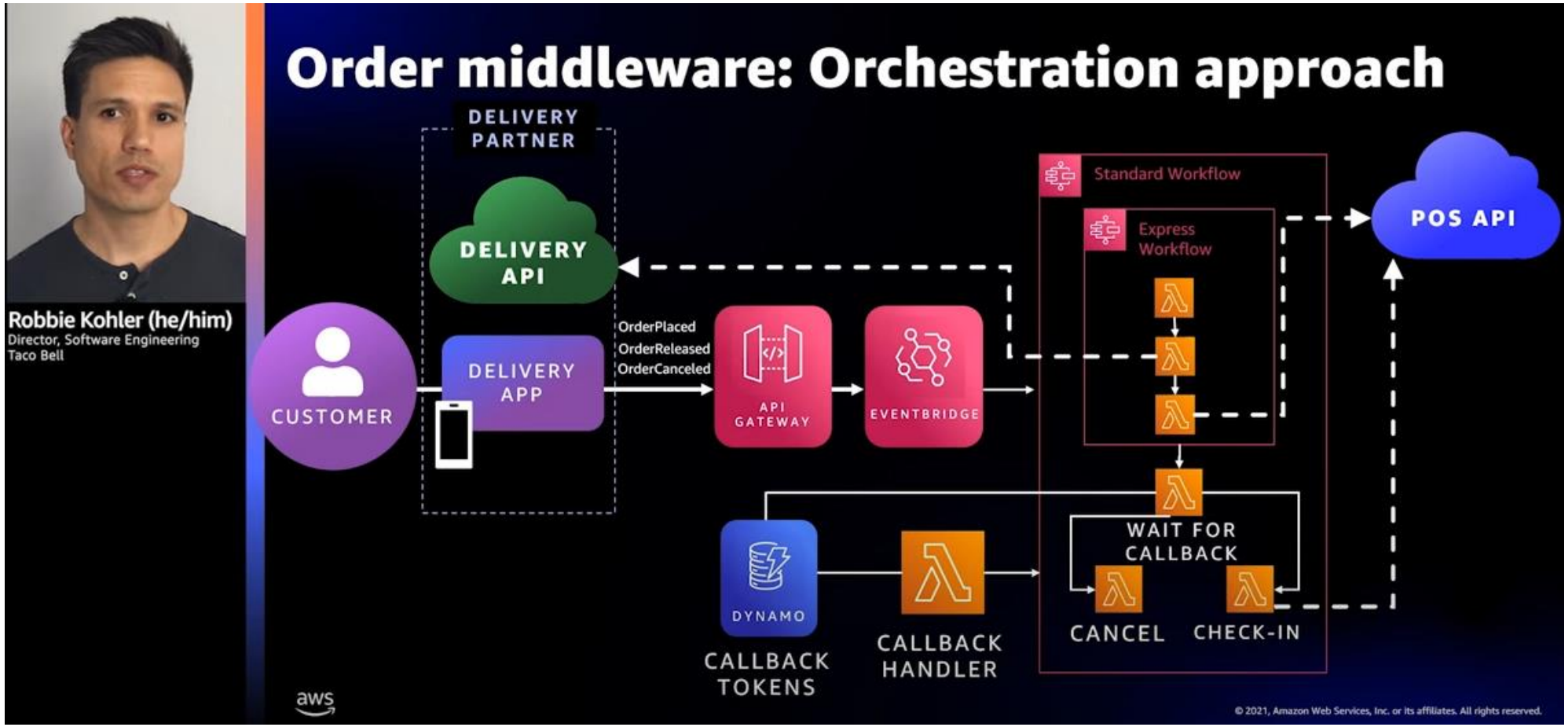
→ What’s Next - AWS Step Functions For Orchestration Order Fulfillment with Serverless Saga Pattern

AWS Step Functions



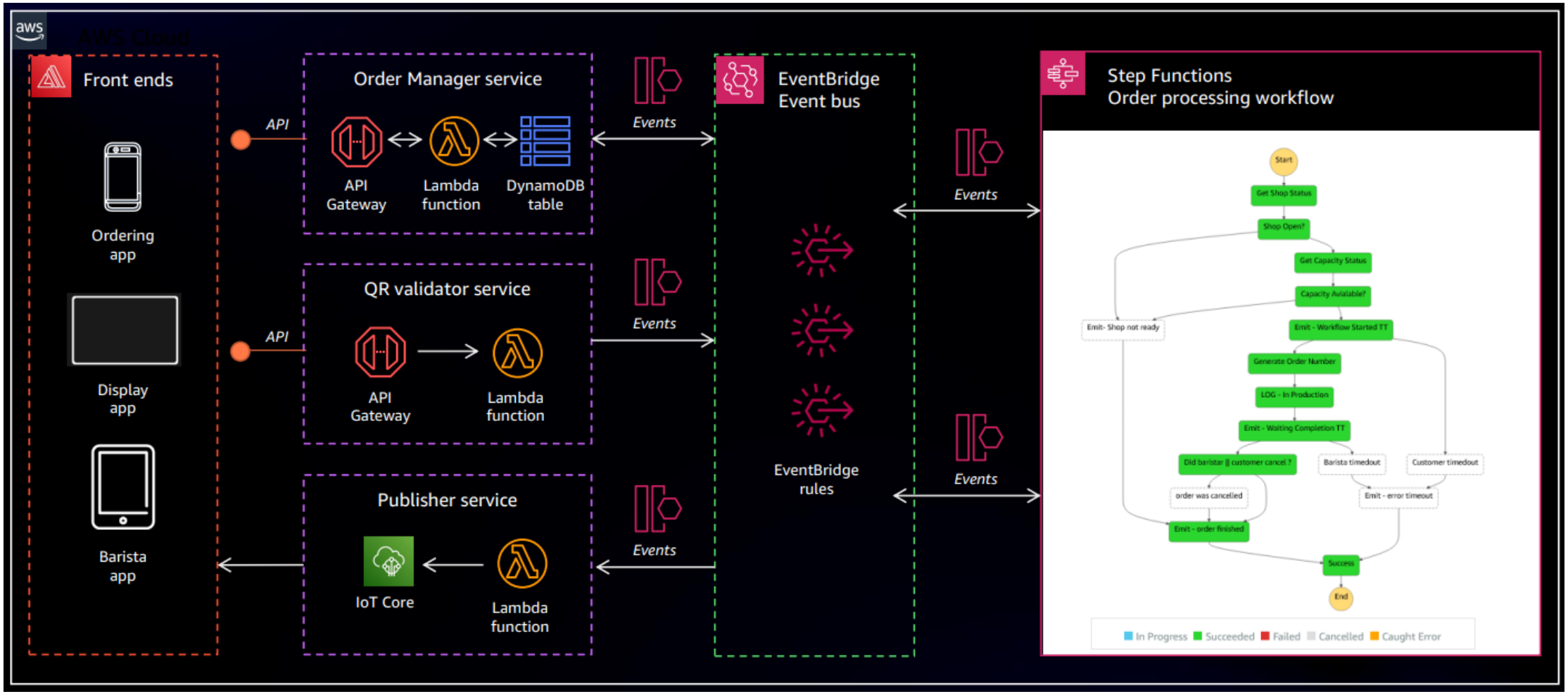
https://aws.amazon.com/blogs/compute/prototyping-at-speed-with-aws-step-functions-new-workflow-studio/

AWS Step Functions For Orchestration Order Fulfillment



https://www.youtube.com/watch?v=U5GZNt0iMZY

AWS Step Function into our E-commerce Architecture



https://da-public-assets.s3.amazonaws.com/serverlessland/pdf/2021+-+Serverlesspresso+exhibit+-+PDF.pdf

Thanks

- Thank you so much for being with me on this journey.

  Reviews and feedback is really encourage to me for pushing forward to create new courses like this.

  Mehmet Ozkaya

Course Logo